UNIVERSITY OF OSLO
Department of Informatics

Temporal Data: Time and Object Databases

Bjørn Skjellaug

Research Report 245

ISBN 82-7368-160-2
ISSN 0806-3036

April 1997

Temporal Data: Time and Object Databases

Bjørn Skjellaug[1]
Institutt for informatikk
Universitetet i Oslo
P.O.Box 1080 Blindern, 0316 Oslo, Norway

bjornsk@ifi.uio.no
http://www.ifi.uio.no/~bjornsk/

Research Report 245
ISBN 82-7368-160-2

April 1997

[1]The author is also part time at Department of Distributed Information Systems, SINTEF Telecom and Informatics, Oslo.

Contents

[1] Named T-OODAPLEX by this author.

[2] Named T-TIGUKAT by this author

Chapter 1

Introduction

> What, then, is time? I know well enough what it is, provided that nobody asks me; but if I am asked what it is and try to explain, I am baffled.
>
> Saint Augustine in the 'Confessions',
> Book XI, Section 14. [8, page 36].

Most computerized information systems are formed from general purpose components, such as databases, GUIs, etc. Whenever properties of, and operations on, these components are generic they are in principle reusable for different purposes and in different contexts. Time and time support may also be regarded as a generic information system component. In fact, most applications have requirements that involve dynamics, and a time-varying nature of both data and processes used by these applications. Temporal databases are means to capture some aspects of the required time support, and are mainly used to manage historic data, present data, and predictive data within the same framework and model. To be more specific, in our context time acts as an intrinsic part of objects denoting when these objects are defined related to the built-in time dimension(s) support of the system. A common name of such objects is temporal objects, i.e., objects capturing, besides other properties, time and time dependencies of their values, e.g. an address history of a person.

1.1 'What, then, is time?'

Time has been a topic of philosophical and natural scientific study since the ancient Greeks (including Plato, Aristotle, and Diodorus Cronus), through Saint Augustine in the Middle Ages and many later Medieval logicians, to Newtonian physics, Einstein's theory of relativity, and on to present [61]. In the second half of our century logicians, computer scientists, and others have

showed significant interest in the task of understanding time in both more formalized, linguistic, mathematical, and technical terms.

The material presented in this report is founded on the above results, in particular, and not surprisingly, mainly of those results utilized and developed by computer scientists.

Time is a human construct. Although it is useful for describing and prescribing changes to the systems and objects under study, as we will see later on, time is in the day to day life most often understood as something which is ever increasing, or is continuously moving in one direction – into the future. This perception of time is more (or only) reflecting human memory. That is, we are able of memorizing the past, e.g. as facts and experiences, but we know nothing about a given time in the future. Still, we may have expectations for and predictions about the future. But, it is only when we reach that time in the future, which in the past was a future time, we can say anything, for example, about the validness of our predictions at the time when the predictions were first stated.

Another model presented by Rucker [73] incorporates time and space into a 4-dimensional space where all "points" are predefined. In such a model we are not "moving" in any specific directions; humans and things are objects defined within the hyperspace. Hence, each 4D object is always showing us when and where we were, are, and (possibly) will be. So, both past, present and future exist simultaneously. Although some points may be regarded as "facts" and others are "predictions", we are within that hyperspace all the time – "going" nowhere – because all entities are defined by their extent in this 4D-space. Rucker elaborates on this 4D view most elegantly in his book: "Geometry, relativity and the fourth dimension" [73].

1.2 Time and databases

In this report we are aiming at an understanding of time as an integral part of properties of data objects. Basically this reduces to understand how properties (or objects) change and behave over time. A common name of such objects is temporal objects, i.e., objects denoting, besides other properties, time and time dependencies. There are different "times", and different notions of time for data management may denote whether phenomena in reality are regarded as *events*, *facts*, or *processes*.

An intuitive interpretation of time objects in a system would be to let events be represented by time points, facts by time periods (or intervals) and processes with time functions. Why these interpretations? To follow this more or less common sense notion of time a bit further: An event is often taken as happening at a time instant, i.e., a point in time with no duration. A fact is a truth about some static aspects of reality but does not last for ever, i.e. a valid-time period defining when the fact was, is, or is

believed to be defined (i.e., valid) in reality. A process like an event denotes or refers to a dynamic aspect of reality, and which, unlike an event, does have a duration. Thus, a process may be a function describing some "cyclic" or repeating behavior like a tide or a continuous behavior like an expanding desert or disease. A process and a fact may for example be defined to last forever, or until terminated or changed, respectively.

We will primarily treat time related to facts because our concern is temporal data and how such data could be used to integrate facts of different kind and/or of different databases. We easily see that if an object changes its value(s) this is in the first place caused by an external/real event triggering some action to do the change (e.g. database update operation). The event itself may have been initiated by a process object. So in a overall setting objects of facts, events and processes are inter-linked. We deal only with the former of these objects, i.e., data management of facts, but will treat the other two whenever they relate to issues discussed.

We also present time and temporal data in a broader sense than is usually the case with temporal databases. We define the notions of schema versioning/evolution, object versioning and configuration management to be part of a broad scope of temporal data. Definitions and examples of these notions are given and we show how they are related to times such as valid times and transaction times, respectively.

For definitions of concepts and glossaries related to temporal databases see Jensen et al. [48].

1.3 About this report

This report is one out of two reports describing the approaches, proposals and other issues within the field of temporal database research. This report deals with temporal object-oriented databases, and the other deals with temporal relational databases [83]. The reports give a survey of their respective subfields, and they classify the proposed models according to structural characteristics and present representatives of every class in more detail. However, equally important is that the fundamental concepts defined by the field is presented, therefore, this introduction chapter and the next chapter are included in both reports to give all potential readers an introduction and hopefully a sound understanding of the fundamental temporal concepts as defined and utilized within the temporal database field.

After fifteen years of active research in the field of temporal databases, about 800 papers published, the first international workshop was held in Arlington, Texas, 1993 [84, 64]. This workshop showed the diversity of issues and topics covered by the field, and the same tendency is documented by the follow-up workshop held in Zurich, Switzerland, 1995 [27, 79]. We are not aiming at covering all the research related to temporal databases, and will,

of course, only refer to those research publications that are relevant for our purpose. The reader who are interested in temporal database bibliographies is referred to Tsotras and Kumar's bibliography [94], and also the electronic bibliography, authored by Kline, containing nearly 1100 entries as of January 1996 [57].

1.3.1 Report Structure

This report is organized as follows: In Chapter 2 we presents some basic concepts that are generic to most temporal database models. Chapter 3 presents a survey of temporal object databases and models, with a classification of the different approaches presented in the literature. The summary, Chapter 4, gives perspectives on both the classification and object-orientation in general in context of temporal databases.

1.3.2 Acknowledgments

I would like to thank Amela Karahasanovic, Ragnar Normann and Dag Sjøberg for comments and suggestions to a previous version of this report.

This research was supported by the Norwegian Research Council through grant MOI.31297.

Chapter 2

Temporal Data — Basics

In this chapter we introduce some generic properties and definitions which are relevant in most settings dealing with temporal data.

2.1 Perception and Construction of Time

A philosophical view of time will probably pin point that an event is not (or necessarily not) an instant of or point in time, and could never be an instant of time because an event happens and something that happens has to have some sort of duration. Otherwise, it would not happen. In natural languages the term event is (in an informal sense) a homonym because it may denote the implicit semantics of time differently: 1) *'the concert was quite an event'*, e.g. meaning an experienced duration of joy and pleasure, and 2) *'at the event he sat down he had a stroke'*, e.g. meaning that something happened at a sudden (or an instant).

So, what is an instant of or point in time? In life and thought we often regard time as continuous say like a river. Thus, a point in time is non-existing or is not an appropriate notion if we want to grasp what continuous time is. However, a point in time is a (mathematical) construct and a notion that gives a sufficient structure of time when it comes to formal reasoning, data management, etc.

With proper models and operations we may sufficiently approximate or describe phenomena by computer systems by time points or other constructs derived from time points, e.g. intervals and periods. Euclidean geometry does the same with natural spatial primitives like for example an extended body that could be approximated by means of theoretical notions like points and lines. That is, the notion of extended body is described in terms of points and lines. See van Benthem [8, chapter I.1] for a more in depth presentation of the above problems, including a discussion of "The Fleeting Now".

2.2 Some basic definitions

Before presenting the different subjects and issues raised and discussed we will clarify some concepts which are used throughout the text (and without any further explanations if not the context defines them separately or relates them to more specific structures). The term *temporal data* means the concept where data or say an object is defined to have some time related information associated with it, e.g. a "built-in" timestamp[1]. For example, in a person object with an property attribute address, and whenever the person changes his/her address the new address value is timestamped. More important, though, is that the previous address values are still accessible, in the sense that an application (or user) may retrieve the whole or a part of the address history of a person. Thus, an address contains both the current and previous addresses (and possibly a future address), as well as the time information associated with each address value. Hence, by means of a temporal data model we may get an integrated knowledge of both where a person lived and when this was. The time (or periods) when a person has addresses is called the lifespan of the address object (see below). The relationship between facts, e.g. a person's addresses, and times is shown in Figure 2.1.

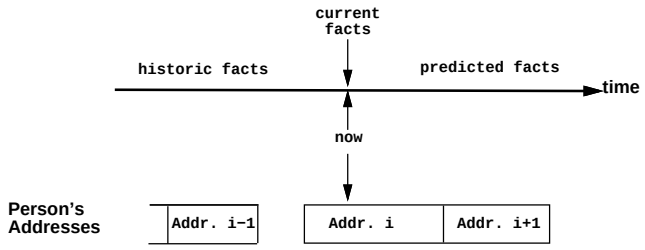

Figure 2.1: Facts and Times

The time information may be given different semantics depending on the time dimensions a particular system supports. For example, temporal database models usually define one or two of the following time dimensions: the *valid-time dimension* (see Section 2.3.1) supports information on when a fact was, is, or is believed to be valid (true) in reality, on the other hand the *transaction-time dimension* (see Section 2.3.2) supports information on when a fact was or is stored/current in the database. The transaction-time dimension has always its upper bound set to the (moving) time *now* and its lower bound is always set to the time when the system was first in operation, i.e., the time when the database was created. Whereas the

[1] A timestamp is a time value (point in time, time period, or time interval) associated with an object [48]. It is the key time representation in temporal databases. But, qualitatively different timestamps represent different notions of time. Hence, an object may have several orthogonal timestamps.

valid-time dimension's upper and lower bounds are set by the application, such bounds for the transaction-time are independent of applications. These two dimensions constitute an object's "real world" history and registration history respectively. In addition several database systems support primitive data types such as DATE, TIME, DATE-TIME, which are used to model what is called user-defined time, and its semantics is defined and only known by the application. For example, an object of type DATE is used to model the property of a person's date of birth, whereas a valid-time timestamp on that persons address property is used to handle the different times a person is associated with different addresses in the modeled reality. Thus, the built-in semantics of valid-time and transaction-time bound objects to the respective dimensions. That is, a time dimension provides more information of being only a value domain, i.e., a time related order of an objects values is maintained and this ordering is utilized, for example, by query processing.

Our concern is valid- and transaction-times and modeling their semantics as part of the database, and not only as structures and semantics of one particular application. Although valid and transaction-times are orthogonal temporal concepts there exist several applications where these two dimensions collapse into one dimension. Many (if not all) real-time database (RTDB) applications have this characteristic. For example, in a cash-line application a bank account is the temporal object of interest. A customer's withdrawal of an amount of money from his/her account means that the registration of this database transactions happens (approximately) at the same (instant of) time as the customers account is changed in reality, i.e. the time when the real money is withdrawn from the account and the time when this withdrawal is registered coincide, and only one (say valid-transaction-)time dimension may suffice for recording both times. Such an database is called degenerated if both valid- and transactions times are captured by one time value [49]. On the other hand when using an ordinary check the times when money is spent and when usage is registered, respectively, could vary form days to weeks, i.e., the valid-time (when spent) and transaction-time (when registered) do not coincide.

Further, a temporal data object is said to have a lifespan, and the time periods or intervals constituting the lifespan are not necessary contiguous. In the address example above, there could be a period when a person does not have a known address. Hence, the address object for this person was not defined for this period of time. Thus, the person had no address (or possibly an unknown address) for a period of time, e.g. see the "gap" between the address values `i-1` and `i` in Figure 2.1. So, the lifespan captures the time when an object is defined. That is, a valid-time lifespan is the time when the object is defined to exist in the modeled reality. On the other hand a transaction-time lifespan is the time when the object is defined to be current and accessible in the database [93, page 625].

We have referred to the concepts *point in time* and *instant* as synonyms.

Another important concept is *chronon*. A *chronon*, which is the shortest duration of time supported, for example, by a database is a non-decomposable unit of time or the smallest granule of time. Based on the notion of a chronon a time period is a set of (total) ordered time values, of which each is separated by a chronon. If a time dimension is discrete and linear the concepts of point in time, time period, and time interval are equivalent constructs for representing time [8]. Although this is true for the semantics of time, one construct may be preferred because applications interpret these semantics in context of the modeled reality. We return to this issue in a later chapter. In most temporal models the chosen time dimension is discrete [58, 62], and therefore it is isomorphic to (some subset of) the integers, or isomorphic to (some subset of) the natural numbers when it has an exact lower bound.

2.3 Time Dimensions and Definitions

Time dimensions are (usually) classified into valid-time and transaction-time dimensions. Both dimensions can be defined separately, and, hence, they are orthogonal.

2.3.1 Valid Time

The valid-time dimension supports not only management of histories and current data, but also planned or predicted data. Histories are data defined for previous times compared to the time *now*. Current data are data valid at the time *now* (i.e., present time), as opposed to predictive data which is data believed to be valid in the future. Figure 2.2 illustrates these properties of valid-time data. Thus, valid-time of some fact is the time when the fact was/is (believed to be) true in reality. Thus, a fact may have several instances, each with an associated timestamp recording changes of that fact [48].

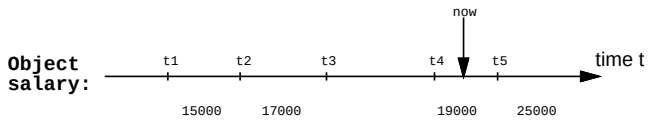

Figure 2.2: Valid-Time Data

Figure 2.2 shows an object salary. It could be an object recording the salary evolution of an employee of a company or a general salary object defining the salary of a particular professional category of a company, e.g. an assistant researcher, an engineer. Despite the difference in possible interpretation of the actual object say by an employee or a professional category application, the temporal semantics of this object remain the same. That is,

the object salary had from time t_1 to time t_2 a value of 15000, from time t_2 to time t_3 a value of 17000 and so on. Some interesting facts are also stored or are part of the semantics of the temporal object salary.

Firstly, between times t_3 and t_4 no value was recorded. We say that the object was not valid (or undefined) during that period. For the forementioned applications this could be interpreted as follows: 1) the actual employee in question had no salary (or even more likely s/he was not an employee) during that period, 2) the company had not a fixed salary for this professional category (or the professional categories did not exist) during that period.

The second interesting fact says something of what is planned or what is believed to be true (or be a fact) in reality, namely that the salary will be raised from 19000 to 25000 at time t_5 and will be fixed until some uncertain (unknown) time in the future. This could be interpreted by an application along the line of the employee and professional category examples above.

And last; the time *now* is in our example currently between times t_4 and t_5. We will discuss the issue concerning *now* and other variables in Section 2.3.4.

We have presented some of the goodies of a valid-time database, and given several examples of the semantics captured with this approach. A nice property with valid-time is that its basic semantics are independent of applications and their interpretations of the data. Thus, the basic temporal semantics are only managed by the built-in valid-time support of the database. This is true even when the valid times are usually given by the user. Hence, we distinguish on one side what valid-time is, and on the other for what it is used (or how it is used) by a user. The former concerns the database semantics the latter concerns some specific application's interpretation of it.

Below we discuss the transaction-time support where both semantics and usage are given and controlled by the database system. Valid- and transaction-times support increases the data independence of databases from applications. This aspect is the main technical argument for extending a database model to a temporal database model.

2.3.2 Transaction Time

The other time support in temporal databases is the so-called transaction-time. A transaction-time timestamp registers the time when an object (or say a fact) is current in the database, and may be retrieved [48]. As with valid-time a fact may have several value instances and timestamps associated with it. Although transaction-time never exceeds the current transaction-time (or time *now*) a fact has also associated with it transaction timestamps for predicted valid-time data when both time dimensions are supported by the database.

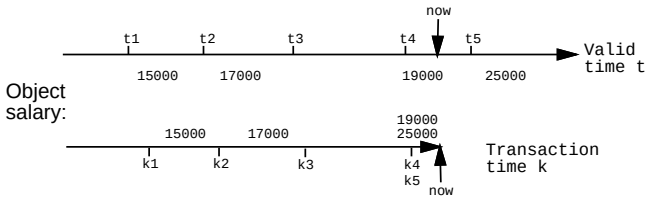

Figure 2.3: Transaction-Time Data

Figure 2.3 shows how the instances of the fact salary are associated with transaction timestamps. At time k_1 the salary value 15000 was first recorded in the database. And it was current in the database until transaction-time k_2. At transaction-time k_3 the fact was logically deleted but restored at transactions-time k_4 with the new value 19000. At the same time there is another transaction-time k_5 (equal to k_4) which records that the fact has value 25000, but not valid before t_5. Both values are current until the time *now*, which is always an upper-bound constraint on transaction timestamps. With transaction-time alone this information makes no sense besides saying something about when a particular value was stored. However, together with the valid-time support it will be interpreted as that both the current salary of 19000 and the planned/predictive salary of 25000 were recorded in the database within the same transaction or by concurrent transactions, respectively, but at the same time. This makes of course sense for both the employee and professional category applications above.

Transaction-times associated with facts have a property or restriction not shared with valid-times. That is, they are not to be altered when first stored. Put differently, we cannot change what already has happened, i.e., at previous transaction times. The transaction-times are in an one-to-one correspondence with the actual database transactions or operations performed, i.e., they register the modifications activities of the database.

2.3.3 Bi- and Multi-Temporal Dimensions

If a database captures both valid-time and transaction-time, we will have the relation shown in Table 2.1. This relation is the representation of the object salary shown in Figure 2.3. For convenience and hopefully to make the illustration more intuitive we think of this relation recording the salary of a professional category, and all symbolic times, t_i and k_j, are replaced with month-year timestamps, still, representing a similar information as above.

A database that supports both valid- and transaction-times is called a *Bi-temporal Database* [28]. In principle there are no limitations of how many time dimensions a database may support, and multi-temporal databases are in fact possible. In other words the 2D time associated with database facts

Salary	*Valid-Time*		*Trans-Time*	
15000	Jan88	Dec89	Apr88	uc^2
17000	Jan90	Mar92	Mar90	*uc*
19000	Sep94	Dec97	Oct94	*uc*
25000	Jan98	∞	Oct94	*uc*

Table 2.1: Object Salary represented as bi-temporal data

in Table 2.1 could easily be extended to capture n dimensions.

Jensen and Snodgrass present application scenarios of multi-temporal databases [49]. The multiple dimensions are based on a taxonomy where a valid timestamp is defined relatively to transaction timestamps. For example, a valid timestamp is restricted not only to one but several transaction timestamps, say when data is defined and stored both locally and globally by several databases. For one valid time there may be several transaction times and these times have restriction imposed on them. All such restrictions could be defined in a schema. A generalization of the multi-temporal nature of [49] is provided by STSQL, a spatio-temporal extension to SQL–92 [11]. In STSQL both multiple valid-times and transactions-times may be defined for a table. That is, multiple valid-time dimensions are associated with an object, where each dimension either denote a separate temporal aspect of an object or a possible world of an object. For example, if we have tasks to be planned and scheduled, one valid timestamp may denote the aspect of when the data stored about a task is true in reality, whereas another valid timestamp may denote the aspect of when the task itself should be executed, i.e., when it is scheduled. If we have possible world semantics, a third valid timestamp would possibly denote an other time when the task should be scheduled, because this time is estimated with some other parameters.

The value-added property of having bi- or multi-temporal relations is that these dimensions capture inter-dimensional semantics which are important for deducing information about the facts stored in the database. It requires, of course, that if such inter-dimensional semantics exist (i.e., their relationships are expressible), each dimension has to be identical, equivalent or compatible with some underlying notions of time. For example, the relation in Table 2.2 illustrates some of the relationships.

The second of Paul's rows, in Table 2.2, represents a so-called *pro-active* change of the database stating in December 1990 (start of trans-time) that Paul will be "re-hired" as an assistant from March 1991 (start of valid-time), i.e., the fact is stored before it is expected to be valid in reality. Paul's has a *retro-active* change of its position (row 4), i.e., in December 1991 the database registers that Paul was an engineer from August the same year. At this time it is also known that the information that Paul was an assistant until November 1991, in fact, was misleading information current

name	*department*	*position*	*Valid-Time*		*Trans-Time*	
Paul	design	assistant	Jan90	Sep90	Feb90	*uc*
Paul	design	assistant	Mar91	Nov91	Dec90	Dec91
Paul	design	assistant	Mar91	May91	Dec91	*uc*
Paul	design	engineer	Aug91	*uc*	Dec91	*uc*
Mary	design	manager	Jan90	Feb91	Jan90	*uc*
Mary	HQ	AD	Dec90	Dec93	Feb91	*uc*

Table 2.2: Bi-temporal Employee Data

in the database until December 1991 (row 3), i.e., Paul was an assistant only until May 1991. That is, all data retrieved before December 1991 may have resulted in inaccurate information. To restore a consistent database regarding Paul's position the following is done: The second row is logically deleted from the relation marked with a deletion time as the upper limit of the transaction timestamp. The information (as best known of December 1991) when Paul was an assistant this year is shown by the third row. The following row captures the present information that Paul is an engineer, and has been so since August 1991. Note that Paul was not an employee during the periods of October 1990 until February 1991, and in June and July of 1991, or Paul had no known position in the company during these two periods.

In the first of Mary's rows the two times coincide for her being a manager, i.e., the database and the reality modeled by it are synchronous in some sense on the fact that Mary was a manager during that period. The last row, together with the previous one, shows that during a specific period Mary held two positions in the company, i.e. from December 1990 until February 1991 Mary was both a manager and an AD.

There are a number of relationships between time dimensions such as those mentioned above. See Jensen and Snodgrass's discussion and taxonomy on temporal specialization and generalization for a more exhaustive presentation of this issue [49].

2.3.4 NOW and other variables

The time *now*, as defined by [26] and [39], in our example in Figure 2.2 is currently between times t_4 and t_5. A function with domain time could return the salary of the object at a particular time-value. In particular, the function *now* will always return the current salary of the object. Notice that the notion of *now*, and its like, makes temporal databases becoming variable databases. That is, we may store a salary say 10000 with a timestamp $[t_i, now]$ as a variable temporal object (not shown in Figure 2.2 and Table 2.1). However, *now* is a (continuously) moving target, so whenever *now* changes the timestamp changes and then the temporal interpretation

of the object changes. The same applies for queries: *"what is Mary's position now"*, which obviously would give different answers when *now* varies. In contrast, what is Mary's position as of May 1992 always yield the same answer.

Thus, it is a semantic difference between models supporting only fixed timestamp(s) and those who support variable timestamp(s). It could be thought of as a difference between extensional and intensional models as Clifford and Isakowitz point out, but without overloading these terms they named them variable databases [28].

2.3.4.1 Variables and Semantics

Alternatives and supplements for *now* are variables such as *uc* – until changed – [97], and distinguished individuals such as ∞[3] [86]. They all denote different semantics, but we will not discuss these issues here. However, it should be apparent that *now*, *uc*, and ∞ by their names denote different semantics.

Although such variables may be intuitively understood and used for the same purpose in different contexts/applications (and in fact they are used that way), we need a formal model to define them precisely. So, if the variable semantics managed by the database are unambiguous at the database level, the interpretation of what a variable denotes in a specific context is a transformation based on the database semantics but done by an application only.

For example, a personnel application and a project application would most presumably interpret the data of Table 2.2 differently because they operate in different contexts. A personnel "user" could interpret the data about Paul being an engineer (i.e. the variable *uc*) that he is still an employee. On the other hand a project "user" could interpret *uc* that he is a suitable candidate for a project. The semantics of *uc* are the same in both cases, but it gives rise to different information.

An interesting discussion on the issues concerning temporal variable databases and their semantics is found in [28].

2.3.4.2 Formal Issues on Variables

The function of *now* have been studied in several fields, for example in logic and philosophy. The pioneering work of A.N. Prior on time, modality and logic in the 1950's [67] and onwards [66] laid the ground for the formal study of *now*. Kamp gives a formal discussion and analysis of the English word *now* [53]. He studied *now*-calculi – $\mathcal{L}(\mathsf{N})$ – and their semantics in which *now* is an operator, and, hence, not a distinct individual. Not to be too technical, though, Kamp's main result is on completeness of the $\mathcal{L}(\mathsf{N})$'s axiom sets;

[3] Intuitively *forever* is the same as, and is also used as a synonym for, the distinguished individual ∞.

Such an axiom set is said to be semantically complete if a "same" calculus – $\mathcal{L}$ – without the *now*-operator has a semantically complete axiom set, and if $\mathcal{L}(\mathsf{N})$'s axiom set is "closely" related to $\mathcal{L}$'s axiom set. This is obtained easily in the propositional case, where every formula containing *now* in $\mathcal{L}(\mathsf{N})$ is proven equivalent to a formula not containing *now* in $\mathcal{L}$. For the predicate calculi the results are stated to be less general because (some) formulae of predicate calculi may not have any equivalent formulae without *now*. Hence, the completeness has to be proved by other means in this case.

Both Rescher and Urquhart [68, pages 35–37], and van Benthem [8, pages 6–7] have another interesting observation concerning the matter of *now* and temporal structures. The observation is by van Benthem stated as a "global property of temporal structures". The property is called 'homogeneity' and its definition imposes that all temporal individuals are formally indistinguishable. van Benthem requires no ontological difference between any time value t_i and for instance the time *now*. He argues that because the role of a/any *point in time* (e.g. *now*) can be played by any temporal individual, so when formalizing temporality the only interesting notions are those that are interesting in the general case. It seems to us that both Rescher and Urquhart, and van Benthem see in certain situations the necessity of a separate notion of for example *now*. Thus we interpret them such that distinguished individuals and their semantics have only interest in a context of an application domain, i.e., where a specific emphasis on such variables and distinguished individuals is required.

2.3.4.3 Databases — Application of Variables

We regard databases as a representative of an application domain of *now* and other distinguished individuals where it is crucial to define the semantics of such notions precisely. The term *individual* is somewhat loosely used here, what it really means is that it is a function value. The importance of a formal semantic is that a database has to respond unambiguously on the meaning of any access to (temporal) data. So, for *now* and other variables, the meaning of them should be uniformly and unambiguously for all access of the database. The database aspects of *now* are treated by Clifford et al. mostly relative to the temporal relational model [24]. The same authors have in particular discussed these issues in context of TSQL2 [25].

Chapter 3

Time and Object Databases

3.1 Introduction

In this chapter we present a survey of temporal object-based or object-oriented databases[1] (TODB). A main difference between research undertaken in temporal relational databases (TRDB) and TODB is that the object-based approaches presented in the literature have been tailored more by specific needs and requirements of different application domains, such as software engineering databases, medical databases, statistical and scientific databases, design databases in CAD, etc. The key reason for this may have to do with the object-based paradigm's ability to model the reality more closely to what it "is" and how it "changes" dynamically according to users decisions compared for instance to what is possible with the relational model in many applications. Therefore time and temporal features have been associated with special needs of these domains. We also see that both historical and versioning object models[2] are two sides of the same coin for these applications. That is, we may informally state that historical data and versioned data are much more related in the developments of temporal object models. However, some proposals of TODB have taken a more general approach and for example extended existing models with generic temporal capabilities. Hence, such proposals follow the line of research that has been the main research tradition in the field of temporal relational models and languages.

We present different approaches to how time is incorporated into object databases. The proposals seem to have different rationales in the way they incorporate time as a separate notion into object models. We define three main classes which they fall into. They are:

- extending an existing model and language(s) with temporal aspects

[1] We will use the terms object, object-based, and object-oriented interchangeably as a qualifier for the models and systems presented.

[2] Informally versioning is a mechanism to define and manage alternatives, variants, and configurations of objects.

at the interface level as user-defined types (with properties and/or functions), thus, without affecting the underlying model and system. These are called Temporal Extensible Object Models.

- extending an existing model and languages with temporal features and built-in semantics, such as time dimension(s), types, operators, query language extensions, temporal query optimizer, temporal indexing, etc. These are called Temporal Extended Object Models

- managing time in conjunctions with specific needs of different application domains. These are called Temporal Application Tailored Object Models .

This classification tries to capture a broad scope of *what* we regard as temporal object models. The three classes are divided accordingly to the rational and objective behind the developed models. Of course, this classification may be debated, but its main purpose is to separate different models based on origin. Because models are developed by different and orthogonal requirements, and decisions are taken on different grounds and falls into different classes, it does not necessarily mean that models or model features proposed exclude each other. That is, features of one model may be compatible or equivalent, or even incorporated with another model of another class without affecting its basic ideas.

The first class is illustrated by for example Wuu and Dayal who show by means of the OODAPLEX datamodel, [98, 99], that this model is capable of managing and handling time-varying information by its existing non-temporal features. That is, an existing non-temporal model manages uniformly non-temporal and temporal data.

In the second class the main idea is to define time as an integral part of the basic object model(s) and language(s). There has been two ways of doing this. The first is to extend an existing object datamodel. The second is to define its own object model based on the most accepted object-oriented characteristics, i.e. a generic object model.

By the third class models are typically tailored to handle some kind of application demands. Even though this kind of models are defined to meet some specific goals the basic characteristics of the models also take into account time in a more general or generic fashion, i.e. classifying versioning as some sort of (implicit) temporal information.

This chapter gives a presentation of proposed models and their languages which characterize the above mentioned classes. This presentation is slightly based on, but differs also from, the presentations of Snodgrass [87], and by Bertino et al. [9, 10]. The former's emphasis is on temporal aspects of temporal object-based (query) languages and to some respect their underlying

models, whereas the latter compares only temporal data model characteristics. Our objective is to classify the rationale for the proposals and to some degree put the previous comparisons together, and in an informal way focus on which characteristics of the models are compatible and/or incompatible. By forcing the latter focus we believe that we are better capable of identifying what is required so we are able to integrate different temporal object-based data models. Jensen et al. [50], and Clifford and Croker [23] have to some degree, though implicitly, set their foci on integration of temporal relational models.

The former [50] defines a *Bi-temporal Conceptual Data Model* (BCDM) which is said to capture the essential semantics of time-varying data. The BCDM is used as a bridge, by means of the notions of *snapshot equivalence* and a defined set of functions, to map relations and operators of existing temporal models into equivalent relations and operators of some specific temporal relational models.

On the other hand, in [23] a more formal approach is applied. It defines the notions of *historical relational completeness*, though, it is stressed that these notions represent a minimal set. It is distinguished between grouped and ungrouped temporal models[3] and completeness for their corresponding query languages, respectively. Completeness is a "metric" to what degree different query languages may be said to be equivalent, i.e. in the sense of having the same expressive power. The difference between completeness in [23] and the relational completeness proposed by Codd [30], is that Codd defines completeness for different languages over the same relational model. Whereas completeness in [23] extends to completeness of different languages over different temporal relational models.

In the following we will be rather technical and very close to the presentation of the proposed models. This is due to the fact that object models do not have any common, formal basic definition like that of the relational model. Although some object-oriented proposal are formally defined they also more or less define their own notation and basic notions. By that they may also have native definitions of the concepts of type, object, class, etc. Still, these concepts capture characteristics that we intuitively would expect an object model to support. Thereby, making the different models comparable and even to some extent compatible, though, not necessarily equivalent in a more strict and formal sense.

[3] An ungrouped model obeys the First Normal Form, 1NF, restriction on relational model, whereas an grouped model, N1NF, does not, i.e. it allows multi-value attributes.

3.2 The Object-Oriented "Paradigm"

Object-Orientation has its origin in the sixties, and from the development of the programming language Simula [31]. However, object-orientation[4] was first widely known from the development of the programming language Smalltalk [42]. At present the ideas from object-orientation are adopted and further developed in many areas of computer science, among where we find object-oriented databases.

A key difference between relational database systems and object database systems is that the former are based on one common, and formally defined, model, namely the relational model by Codd [29]. Object databases have no such commonly defined model of fundamental notions and concepts. That is, each object database model defines its own basic object model. Moreover, the vast set of object models vary in the number of concepts supported, and to what degree the concepts are formally defined.

However, there have been attempts to define a consensus among researchers and developers on what an object database system should support, see for example [7, 15, 16]. The first attempt was the 'Object-Oriented Database Manifesto' by Atkinson et al. [7], where they describe 8 object-oriented rules. That is, if a database system supports complex objects, object identity (OID), encapsulation, types or classes, inheritance, dynamic binding, extensibility, and computational completeness, then it can be called object-oriented. In addition also more generic database requirements are included, such as persistence, disc administration, concurrency, recovery, and ad-hoc queries.

There are efforts that are concerned with unifying these concepts in standard object models. The ongoing standardization on SQL, called SQL3 [60, 59], is an attempt to incorporate both the relational and objects-oriented "worlds" into one model. The SQL3 efforts have to be considered as a long term project, where some of the already announced features most likely are postponed to a 'SQL4' standard, e.g. fully user-defined object types and encapsulation. A more interesting project, though, is that of the Object Database Management Group, ODMG [15]. ODMG is a consortium founded among object database vendors and other interesting parties, e.g. people involved in standardization[5]. Their strategy is to define a (mature) object model and languages, and submit their definitions to the standards bodies, such as ISO and ANSI, without following the cumbersome and slow development procedures within these standards organizations. However, the

[4] The term *object-oriented* was coined by Alan Kay in the Smalltalk-project, even though the Simula team used the term *object* in the general sense known today, and as it was adopted by Kay in the seventies.

[5] ODMG has an objective not only to unify and harmonize the concepts and notions of object-oriented databases, ODMG should also utilize portability of database schemas, and thereby increase interoperability among object database systems.

success of ODMG has still to be proven[6].

In the following we use the ODMG-93 standard as an example of what an object database model is.

3.2.1 ODMG's Object Model

The main principle is that the basic modeling primitive is the *object*. An object has a *type* that defines its interface, an interface defines the characteristics of objects of a type, i.e. the kind of *operations* and the kind of *properties* defined for the type. Properties are defined either as *attributes* of the object, or as *relationships* between the object and other objects. The operations and the properties capture the possible behavior and states of an object, respectively. Properties can either be defined to be type properties or instance properties, where the former denotes values shared by all objects of the type, and the latter denotes values of distinct objects. Thereby, types are themselves objects.

ODMG supports inheritance as a *super-* and *subtype* hierarchy, i.e. a subtype inherits all definitions of its supertype(s). ODMG defines what is denoted as the *extent* of a type. An extent imposes that a database system maintains the set of instances of a type. The extent declaration of a type is optional. ODMG also supports a *type-class* dichotomy where a type is the specification and a *class* is a implementation of the type, and a type may be implemented by several classes. The class concept is defined both to implement the common behavior and range of states of a type, and maintain the set of objects of that class. In that sense a class realizes the notions of type and extent.

The set of object types supported by ODMG is divided into mutable and immutable object types. Instances of the former type could be created, modified, and later destroyed, instances of the latter are not allowed to be modified. ODMG denotes these as *Objects* and *Literals*, respectively, where objects are identified by their unique OIDs, and literals are identified by their values. Both object types and literal types are subtypes of the type *Denotable_Object*. Moreover, subtypes of object types or literal types are either *atomic* or *structured*. Atomic object types are defined by an arbitrary set of used-defined characteristics, and correspond to the notion of an abstract data type (ADT). Atomic literal types are typical *integers, float, character*, and *boolean*. Structured object or literal types are either *collections* or *structures*. Collections are of type *set, bag, list* (e.g. *string* or *bit_string*), and *array*, where the elements of each collections are of the same type. Structures are similar to records or tuples, where a structure has a fixed number of slots of name-type pairs. Access to a slot is through its name. All of the

[6]The present Object Database Standard, ODMG-93, was published in 1993. A Release 2.0 is expected sometime in 1997.

above mentioned types are built-in types of the ODMG object model, and each type has a set of predefined properties and/or operations.

3.2.2 ODMG Languages

ODMG has, besides the object model, defined three languages; namely an object definition language (ODL), an object query language (OQL), and finally an object manipulation language (OML). Both ODL and OQL are separately defined languages that supports the ODMG object model. The OML is solely defined through and by the programming language bindings specified for ODMG[7].

ODL is the language used to specify the interfaces of the object types[8]. It is programming-language independent and could be used to specify the database-schema semantics for a vast set of ODMG compliant database systems.

OQL is, as SQL, a declarative query language, and similar to SQL in syntax. Where SQL deals with sets as relations, OQL deals with with sets of objects, but unlike SQL, it extends its primitives to include structures and lists. Since operations are part of the characteristics of an object type, operations are allowed to be applied to objects in a query. Of course with the drawback that the query engine looses control of the execution until the operation terminates. OQL does not support a update mechanism like those found in relational database systems and SQL-derivates. ODMG-93 let the native database system language deal with issues concerning database updates.

Neither the ODL nor the OQL are computational complete, i.e. having the same ability to express computations as programming languages. But, the ODMG's OML gives in some sense a computational complete database model. The OML is defined by language bindings to C++ and Smalltalk. Both for ODL and OQL C++ and Smalltalk mappings exist. Hence, OML is nothing else than ordinary C++ or Smalltalk expressions and commands, e.g. conditionals, calculations, and updates, that manipulate structures defined in ODL and/or OQL.

A binding to a particular (programming) language also includes the relationship between the concepts of type and a class in ODMG. The target language implements the interfaces in the sense that the declarations are mapped to constructs in the target language. For example ODL operation signatures are mapped to member function declarations in C++ header files, and their corresponding bodies are implemented in C++ source files.

[7] The language bindings defined in ODMG-93 [15] are for C++ and to some degree for Smalltalk.

[8] ODL is also said to be compatible with the OMG's (Object Management Group) interface definition language, IDL.

3.3 Structure of the Presentation of Models

Following this section we first present models within each of the classes Temporal Extensible Object Models and Temporal Extended Object Models. Then, at the end of each presentation we schematically compare the (temporal) characteristics of the models presented. The comparison has the following structure:

Time Structure: Indicates what the underlying time model is; if it is linear, branching, or cyclic, and discrete, dense, or continuous; or is it user-defined. For example, linear and discrete is isomorphic to the integers, branching and dense is isomorphic to some partial order of the relational numbers.

Time Dimension: Here usually three options exist, if the model supports either valid time or transaction time, bi- or multi-temporal model, or the time dimension(s) is (are) user-defined.

1. Class Objects: Lists the model constructs with unique system generated and maintained object identifiers (OIDs).

Temporal Attribute Values: Indicates what a temporal attribute is denoted by. Usually there are two options

Versioning On: Lists the model constructs which is defined to be temporal. Usually attributes are the construct defined temporal.

Lifespan: To indicate if an object has the ability to be created, destroyed, and subsequently be reincarnated for a specific type. If lifespan is contiguous an object becomes an instance of the supertype of the type it was an instance of when removed from its extent. Models which support a generic object type that all other types are subtypes of usually support a contiguous lifespans, i.e. an object remains an instance of the generic type until explicitly deleted.

Intent & Extent: Indicates what notions the model support to handle the relationship between a specification (intent) and the implementation of the specification (extent), e.g. a type-class dichotomy.

Inclusion & Substitutability semantics: Inclusion semantics means that if T is a subtype of type T', then extent of T constitute a subset of extent of T'. Substitutability means that an operation defined on an supertype, can be applied to instances of its subtypes.

Query Language: Indicates if a model has a defined query language, and list the name(s).

3.4 Temporal Extensible Object Models

Wuu and Dayal's extensions to OODAPLEX [98, 99], Goralwalla and Özsu's extensions to TIGUKAT [43], and Cheng and Gadia's use of OODAPLEX [18] are, as far as we know, the only representatives of this approach where both the underlying model and query language are unchanged. Furthermore, all

their extensions are independent of any specific application requirements.

3.4.1 T-OODAPLEX[9]

T-OODAPLEX is based on the object-oriented data model OODAPLEX [32]. OODAPLEX is an extension of the DAPLEX functional data model [82]. It supports the main characteristics of an object-oriented data model (i.e. object identity, encapsulation through abstract data types, inheritance, polymorphism, etc.).

The main idea behind the temporal extension of OODAPLEX, [98, 99], is to show that time-varying information may be modeled by means of the existing features of the non-temporal data model OODAPLEX. Thereby, Wuu and Dayal argue that the T-OODAPLEX datamodel is a uniform datamodel both for temporal and non-temporal information. There is no doubt that on the conceptual level T-OODAPLEX represents a uniform model both for temporal and non-temporal data. However, the more representational and access oriented issues concerning temporal data are by no means met by this approach. That is, such issues are related to special treatment of temporal data in query optimization, storage and access mechanisms (e.g. paging, temporal indexing) etc.

The basic characteristics of T-OODAPLEX relays heavily on the notions of both the functional language paradigm, parameterized types and user-defined types. The time dimension in T-OODAPLEX is user-defined, and, hence, no OODAPLEX intrinsic time functions are available besides those defined by the designers of a particular T-OODAPLEX datamodel. Time is introduced by an abstract data type (ADT), *point*. This type is the generic supertype of all time types. The rationale for a *point* type is that by the notion of an abstract point, then, for example, events in reality and time instants related to a particular clock or calendar may be treated uniformly on a conceptual level. Hence, the notion of ordering is still relevant to both time-varying information, i.e., an ordering that is associated with events, versions and/or dates. That is, all time types in a particular T-OODAPLEX model are of type *point*, and for each *point* time type, i.e. subtypes of type *point*, a set type is defined through the parameterized type mechanism of OODAPLEX. The $\{point\}$ is the corresponding point set of the type *point*, and $\{point\}$ is the supertype of all point sets defined. That is, the predefined *set structure* type of OODAPLEX could be parameterized with a particular (user-defined) *point* type, T, and the elements of such a set are all of type T.

This generality of abstract (time) point objects and point sets separates the issues of modeling time-varying information from the issues of given these abstract objects a concrete representation. So, implementors have the

[9] Named T-OODAPLEX by this author.

responsibility to define the appropriate time representation of an application, such as a particular ordering (total vs. partial), discrete vs. dense time, linear vs. branching time, time point vs. intervals, which time dimension to support, and the metric information of the time elements supported.

T-OODAPLEX represents more a framework or a tool-set for defining a temporal model rather than being a temporal model it self. From this framework other designers can work out their own basic and specialized set of temporal types and operators, including those characteristics inherited from the framework.

The type *point* defines the operators equality ($=$) between elements, and order ($<$) among elements. These operators may be overridden and redefined by its subtypes and their corresponding implementations, respectively. The point set type, $\{point\}$, defines operators such as $\in$,$=$, $\cup$, $\cap$, $\subset$ on sets. In the same manner as for type *point* these operators may be overridden and redefined to meet needs and requirements of more specialized types.

Because of the functional nature of the OODAPLEX model all properties of an object are specified as functions of its corresponding type. But, there is a distinction between functions which are of the temporal sort and those of a property sort. The former sort has always the time dimension as its domain, the latter sort has the set of OIDs of a particular object type. Functions are also first-class objects with *function type* denoted as $(D \rightarrow R)$, where D is the *domain type* and R is the *range type*. That is, a function instance is a partial mapping from $extent(D)$ to $extent(R)$[10]. In particular a property function takes an OID as its input and return a temporal function that maps a time value into a snapshot value of the property. In this case the property function type is denoted as $(D \rightarrow (T \rightarrow R))$, where D is its domain and $(T \rightarrow R)$ is its range. T is the time domain of the temporal function involved. Each temporal property of an object has a so-called temporal element associated with it. A temporal element gives the *lifespan* of an attribute, i.e. it tells when this attribute was/is valid in the modeled reality.

```
type employee is person
     function name (e:  employee → n:  string)
     function salary (e:  employee → f: (t:  time → m:  money))
```

T-OODAPLEX allows properties of type temporal, immutable and non-temporal (static) to be defined for an object type, but all properties is defined as functions. This is shown above with the `name` property as a static property. The immutable type `string` is a basic data type, and, hence its representation is pre-given and ever lasting. Although a string value associated with `name` may of course be deleted or overwritten in a particular object, the value

[10] In the T-OODAPLEX case the function $t\_extent(P)(t)$ returns the instances of type P at time t, and the mapping could be constrained to this function if temporal. The OODAPLEX $extent(P)$ will return all instances of P at all times. Still, inclusion semantics is retained in both cases.

itself is always an element in the domain of type `string`. The temporal property `salary` has its range defined as a temporal function which maps a time element into snapshot values of the property.

The above presentation shows how properties or say attributes values are made temporal. T-OODAPLEX defines timestamps on objects as well. That is, not only attributes are versions also objects are. The typical way of doing it is to choose an acyclic graph structure of nodes where each node represents an object version. In the same way as all time types are subtypes of type *point*, so are a version type because versions are also represented by discrete points, e.g. evolution of a design and releases of a software product. An object captures a structure of its own versions. This structure imposes parent-child and sibling relationships among object versions, e.g. in design applications it would typically be modeled as relationships between the previous and next design versions or as design alternatives respectively. The (implicit) time of object versioning gives a branching time, because alternative object versions may obviously exist simultaneously in time, i.e. an object version graph imposes a partial order.

3.4.2 T-TIGUKAT[11]

TIGUKAT is called an "object-base management system" [63], and its data model has much in common with the functional nature of OODAPLEX. Still, TIGUKAT has a clear distinction between the notions of type and class. A type defines object characteristics as a behavioral description (interface), whereas a class is used for the implementation of a type, and management and maintenance of instances of that type. In principle, a type may be implemented by several classes. T-TIGUKAT's data model [43] differs also from T-OODAPLEX as follows: T-TIGUKAT does not differentiate between attribute and object versioning, which are handled in a uniform manner. Object histories[12] are not only restricted to an object being a member of a class, it also extends to include an objects membership in any collection during its lifetime or at some particular time. Temporal (attribute) values were in T-OODAPLEX denoted by functions, in T-TIGUKAT such values are denoted by sets of pairs (t, o), where t is given by one of the optional user-defined time-models, and o is the object instance as it is at time t. Because T-TIGUKAT has user-defined time-structure it could be of any combination of linear or branching with discrete, dense, or continuous. The type system of abstract data types is defined to handle these notions of time, as well as explicit behavioral types which specify how to handle temporal behavior of objects. With the strong emphasis on modeling behavior all properties of objects are modeled as behavior. A particular behavior defines, in technical terms, the semantics of an operation. For example if Mary is an employee

[11] Named T-TIGUKAT by this author

[12] In T-TIGUKAT this is named behavior histories and/or lifetime behavior.

(i.e. in T-TIGUKAT terms an object o_i named 'Mary' is a member of the class `employee`), and `salary` is defined as a temporal behavior of the corresponding `employee` type (i.e. interface), then `salary.history`(o_i) returns Mary's salary history. The operation `history()` is inherited from the behavior type mentioned above, and because Mary's `salary` is defined to be temporal. All behavior is defined as subtypes of a generic behavioral type. For temporal behavior a specific subtype of behavior is introduced which specifies the generic behavior of all temporal types. Functions are used to implement such behavior. By the type and behavior notions for specification, and class and function notions for implementing type and behavior, respectively, all data are encapsulated by TIGUKAT.

In [43] which first defines the temporal extension of TIGUKAT there are no clear definition if the supported time dimension is only valid time. Whereas is [63] it is stated that both valid and transaction times may be supported, and even both may be supported simultaneously as for a bitemporal database.

The TIGUKAT's query language, TQL [43, 63], also used for T-TIGUKAT, is an extension of the basic TIGUKAT object model. That is, queries are defined by means of types and behavior as subtypes of the type query. In this way queries are regarded as first-class objects, and, hence managed by the system as objects. Without going into details, the query language is based on an object calculus with an equivalent object algebra. The user language is said to be similar to that of SQL3.

.

3.4.3 OOTempDBM

Even though Cheng and Gadia [18] also use OODAPLEX as a basis for their temporal object model, OOTempDBM, they do not use the query facility of OODAPLEX, but rather an object-based query language called OOTempSQL which is based on TempSQL. TempSQL is in turn a temporal extension of SQL, and it includes new temporal operators and constructs for access of temporal data.

The OOTempDBM is an object-based extension of Gadia's homogeneous relational model [39]. That is, the homogeneity assumption is enforced all temporal attributes of an object[13]. Moreover, all types need to have a key (in the relational sense), but if a key is not explicitly specified the type inherits the key of its supertype(s). Mainly because of this explicit key restriction (which is used for some undefined kind of relationship between extents related with types) it seems that they misinterpret both the semantics of being an instance of some type and that of an instance having a unique object identifier (OID) in an object based system. Or they mix conceptual issues

[13] Homogeneity constraints all temporal attribute values of each state of an object to have identical timestamps [39].

with those issues concerning the representation of the data. To be more specific; In general an instance of a type is by definition also regarded as a member of all its supertypes. Let object o be an instance of type T, then in OODAPLEX terms o is an element of the *extent* of type T, i.e. $o \in extent(T)$[14]. If T is a subtype of type S, then $o \in extent(S)$ as well, and consequently $extent(T) \subseteq extent(S)$. However, in OOTempDBM this is not explicitly defined by the model, e.g. by using their example [18, page N-11], an employee type is a subtype of type person, but an employee instance named 'John' has a different OID than the person instance named 'John'. Both instances model the same real world entity named 'John'. In object databases, because of the subtype relationship, we conceptually expect that the database object corresponding to the real world entity 'John' is managed only as one logical object with one unique OID. In OOTempDBM it seems that this conceptual information is managed by several objects and OIDs, and the application has to keep track at any time instant of whether 'John' is regarded as both a person and an employee. This information is only implicitly given by the model. The OID mechanism supported by OOTempDBM is better compared with some (system internal) foreign key mechanism like that of a relational system.

Like T-OODAPLEX, properties are of type temporal, immutable or non-temporal (static), and their temporal values are functions.

3.4.4 Models Characteristics and Summary

Table 3.1 lists some of the main characteristics of the three proposals.

T-OODAPLEX [99] and TIGUKAT [43] both explicitly define a set of basic temporal constraints which should be enforced by the systems, e.g. a temporal inclusion semantics constraint. In the case of T-OODAPLEX; Important functions for defining the temporal constraint are the functions of *lifespan, extent*, and *t_ extent*. A *lifespan* is telling either when an object is valid (exists), a type is defined, or a database is operative. An *extent* of a type tells which objects are instances, and which are members[15] of this type. A *t_ extent* does the same but at a specified time. The constraints define temporal extensions to the inclusion semantics, substitutability, and referential integrity, etc. OOTempDBM has only a limited, informal descriptions of its temporal constraints.

[14] OODAPLEX does not support a type-class "dichotomy", where a type is the specification of the intent (interface), and a class is the implementation of some type(s) and acts as a placeholder of all object instances of that class. Hence, a type may be implemented by several, possibly non-overlapping, classes. In OODAPLEX there is only a one to one correspondence between a type T and its extent $extent(T)$

[15] A member is an instance of a subtype of a type.

Model Charact.	**OODAPLEX**	**OOTempDBM**	**TIGUKAT**
Time Structure	user defined	linear & discrete	user defined
Time Dimension	arbitrary[1]	valid	valid & transaction
1.class objects	objects functions[2]	objects functions[2]	objects[3]
Temp. attr. value	function	function	timestamp-value pair
Versioning on	attributes & objects	attributes	arbitrary[4]
Lifespan	contiguous	non-contiguous	non-contiguous
Intent & Extent	type & extent	type & extent	type & class
Inclusion & Substitute.	yes	no(?)	yes
Query Language	OODAPLEX	OOTempSQL	TQL

[1] Valid or transaction time. Or both denoted by $f : (D \rightarrow (T_v \times T_t \rightarrow R))$, a function type.

[2] Since function types is supported functions are first class citizens.

[3] Unclear if both functions and collections are 1.class objects

[4] No distinction is made between attribute and object versioning.

Table 3.1: A brief summary of OODAPLEX, TIGUKAT, and OOTempDBM

Other Temporal Extensible Object Models In this class of temporal object models we could include for example other models like TMAD (and MAD) by Käfer & Schöning [52, 51], and VISION by Caruso and Sciore [13, 14]. Although both TMAD and VISION support temporal features and principles of generic nature, we choose to present these proposals in Section 3.6 because their developments are more application driven.

3.5 Temporal Extended Object Models

This class of temporal object models incorporate time as an integral part of the basic object model itself. Two main approaches has been proposed; the first is to choose an existing model and extend it with temporal capabilities, the second is to define a time model and integrate it into a generic object model that support the most expected notions of an object model. We present candidates of both alternatives.

Of the former alternative we choose to present the $T_$Chimera[16] model by Bertino et al. [9, 10], and for the latter we present the TOODM by Rose

[16] The $T\_$ stands for Temporal.

and Segev [69].

3.5.1 $T\_$ Chimera

The $T_$Chimera model [9, 10], is an extension of the non-temporal object-oriented data model Chimera[17] [44].

The type system of the $T_$Chimera model is where the temporal aspects are introduced. The $T_$Chimera model inherits all types of the Chimera model. First; the set of all Chimera types, $\mathcal{CT}$, is the union of all literal types (basic and structured types) and the set of all user defined types, called Object Types, $\mathcal{OT}$. The set of $T_$Chimera temporal types, $\mathcal{TT}$, is simply introduced by letting every $T \in \mathcal{CT}$ be denoted as *temporal(T)*. Then the set of all $T_$Chimera types, $\mathcal{T}$, is the union of $\mathcal{CT}$, $\mathcal{TT}$, structured types of temporal types (e.g. *set-of*(T), where $T \in \mathcal{TT}$), and the singleton set *time* type. These are formal prerequisites of the type structures that extend the model of Chimera, i.e. a structural extension by which the temporal information is managed and maintained.

The interpretation of the *time* type is that it is isomorphic to the set of natural numbers; $\mathcal{TIME} = \{0, 1, 2, ..., now, ...\}$, where 0 is the relative beginning of time and *now* is the current time. Hence, the time structure is discrete and linear with only the valid time dimension supported. Timestamp are associated with attributes of objects. Even though both classes and instances of classes, i.e. objects, have *lifespans* they are by no means regarded as timestamps in the conventional sense[18]. A timestamp is said to be defined only on attributes. In the case of a class, if any of its class attributes[19] are of type *temporal*(T), where $T \in \mathcal{T}$, then the class is regarded as *historical*. Otherwise, the class is *static* and records no time-varying information of its own class attributes. Objects are also time-varying, if any of its instance attributes are of type *temporal*(T), where $T \in \mathcal{T}$.

On the other hand, lifespans of both classes and objects are defined to be the time when they exist, and capture the temporality of these notions. $T_$Chimera also indicates that both classes and objects are constructed (and possibly destructed) in the sense that a lifespan has always an absolute beginning (and possibly an absolute ending, respectively). A direct consequence of this is that $T_$Chimera does not support a *reincarnate* function. Therefore, a *lifespan* of a class or of an object is a set of contiguous time instants. $T_$Chimera only supports valid time, and lifespans are used for different consistency considerations, for example, on attributes. Thus, it is clear that

[17] The Chimera data model is originally an object-oriented, deductive and active data model, but features such as triggers and deductive rules are not extended with time in this version of the $T_$Chimera model [10].

[18] ($T\_$)Chimera regards also a class as a first class object

[19] A class attribute is analogous both to a class variable in Smalltalk [42] and a public static variable in C++ [90], and is shared by all instances and members of its class.

a class (or object) lifespan reflects when a class (or an object) is valid in the modeled reality.

Furthermore, according to a type hierarchy an object may belong to different classes simultaneously, but it is an instance of only one class at any one instance $t \in \mathcal{TIME}$. This is called the object history of an object. For example, take a personnel and project database where employee objects associated with projects are also defined as project member objects. For simplicity, we regard this as a type hierarchy in $T_$Chimera, where `projectMember` is a subtype of the type `employee`, i.e. conceptually representing employees who are specialized to work on projects. When an employee becomes a project member, the corresponding object becomes an *instance* of the class `projectMember` and only a *member* of its superclass, i.e., the class `employee`. If the same person leaves the project(s), but, still, is regarded as an employee, the corresponding object instance again becomes an instance of type `employee` and is logically deleted as an instance of type `projectMember`. We say that an object may migrate between classes according to a type hierarchy. Hence, the object's set of properties change accordingly to which class it is an instance of – at any given point in time. This is the temporal information that is managed by the notions of class and object in $T_$Chimera, i.e. in practice by their internal property slots called *history* and *class-history*, respectively. The slot *history* manages at all valid times all objects that are proper elements (i.e. instances) of a class extent and that are members according to their subtype relationships. On the other hand the slot *class-history* manages at all valid times all the class an object is an instance of, i.e. a proper element of.

The $T_$Chimera model has three different concepts of modeling time, i.e. temporal attributes, lifespans of objects and classes, and object histories. All three are important for the definitions of $T_$Chimera's notions of consistency. The notion of lifespan, and consistency based on that notion, is similar to that of T-OODAPLEX [99], and is used to define constraints .

3.5.2 TOODM

The TOODM model by Rose and Segev [69] is not based of any existing object model, but rather it incorporates a time model in a generic object model. A generic object model supports at least a minimal set of notions that is expected to classify it as an object(-oriented) model. In TOODM these notions are multiple inheritance, encapsulation, and object identity. These are said to give rise to primary language constructors such as object, class, type, method, message, and collection.

TOODM defines a basic type lattice (a directed acyclic graph, DAG), where all types are subtypes of the type OBJECT. The OBJECT type has three major subtypes; Class, PTypes, and Collections, where subtypes of these types are typically abstract data types, primitive types (*integer,*

boolean, time, ...), and structured types (*set, tuple, sequence*,...), respectively. The type lattice is shown in Figure 3.1.

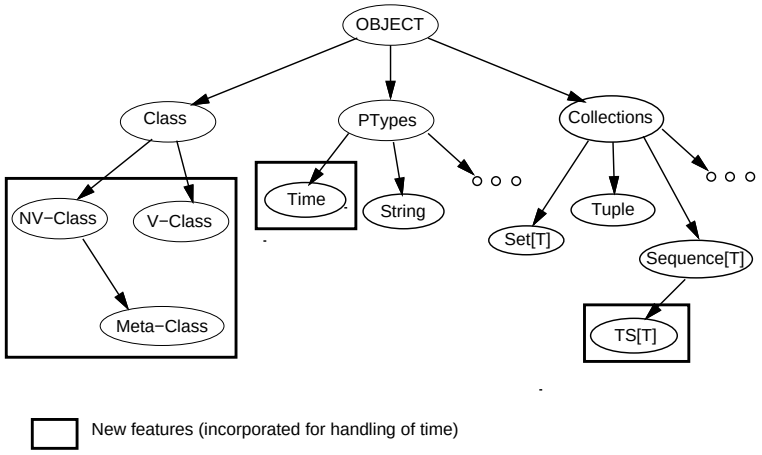

Figure 3.1: TOODM's System Type-Lattice – figure 2 in [69].

User-defined types are defined as subtypes of the predefined types. They are either defined as subclasses of type Class[20] (i.e. subtypes of Class or its subordinate types), subtypes of PTypes, or as subtypes of type Collection. In the latter case a user-defined type could be a parameterized type of types $set(T)$, $sequence(T)$, etc. where T is a legal TOODM type of any kind, or a *tuple* of the form $\langle a_1 : T_1, a_2 : T_2, a_n : T_n \rangle$ where each a_i is a field name and each T_i is its legal TOODM type.

The temporal type extensions of the basic data model is shown within rectangles of Figure 3.1. The NV-Class and V-Class are used for non-versional and versional classes, i.e. the type definitions are fixed and time-varying, respectively. The Meta-Class definition is included so that different types (i.e. classes) could be treated as instances of a meta type. One application of this Meta-Class notion is to rename types. That is, different users may assign different names to the same object type at different times.

TOODM introduce a PType called *time*. All time types are of this type or of its legal (user-defined) subtypes. The time structure is not well defined, but at one point it is stated that "Time in TOODM, is viewed as continuous and independent of events which are defined as duration-less happenings..." [69]. By no further explanation of what they mean with 'continuous', still, they claim they adopt the taxonomy of time for valid and transaction times presented by Snodgrass & Ahn [85]. In [85] (and also in [69]) time appears in the examples as discrete time points or intervals with a discrete

[20] TOODM does not clearly distinct between the notions of types and classes. That is, the term class is used to define abstract data types as well as implementing those types.

starting time and possible the variable *now* as the end time. Therefore, we assume that the time structure is discrete. Further, time can either be absolute like time points or intervals, or it could be relative as for periods with fixed (or a possible non-fixed) duration. TOODM does not only support previous and current times (i.e. for historical data), also future time points is supported (i.e. for predictive data). If data is predictive, then the time structure could be branching from time *now* and into future times. If it is linear, then only one prediction per data object is applicable, which may make sense in some applications, but not in others.

TOODM supports (at least) three time dimensions; valid time, transaction time, and event time. The latter is defined to define the time when an event occurs. For example, an event time could be the date when a manager decides to rise the salary of an employee. This date may be different from the date from which the new salary is valid, which may be a date either before or after the event time. Timestamps are represented as either time points or time intervals, where the former seems to be mandatory for event time.

All temporal information of data objects is managed by defining attributes of abstract data types (i.e. classes) as time sequences ($TS[T]$, where T is a legal TOODM type). For example a temporal/historical salary attribute (`salary-h`) of an employee object is defined as `salary-h: TS[Salary]`. The time sequence type, $TS[T]$, is pre-defined and is given by a 6-tuple: `(name, surrogate, history, predicates, order, methods)`, where `name` is the name of the type, `surrogate` is an identification (i.e. OID), `history` is a pair (list of attributes, list of time lines[21]), `predicates` is a set of constraints imposed on the attributes and time lines defined by `history`, `order` defines an ordering of the history elements (i.e. identifies the time line that defines the order), and `methods` is the set of operations applied on objects of this type. It is the `history` component of the time sequence type which gives rise for management of temporal information. This component is analogous to a temporal relation definition with tuple time stamping.

An instance of a time sequence type, i.e. a time sequence object, is a triplet `(OID, history, corrections)`. The `OID` is the identification of the time sequence object, `history` and `correction` components represent each a set of 3-tuples, where each tuple contain a history OID, an actual (attribute) value(s), and a temporal element. The history OID is used so that tuples in the history and correction sets, respectively, may reference each other so that if the application has changed a (historic) value its new value is found in the correction set. The temporal element has as its domain the cross product of time lines supported, i.e. as defined by the `history` component of the corresponding time sequence type. For example, if valid, transaction, and event times are defined, then $(T_v \times T_t \times T_e)$ defines the domain of the

[21] A time line is the same as a time dimension

temporal element; where, according to [69], intervals are valid timestamps of T_v, intervals or instants are transaction timestamps of T_t, and instants are event timestamps of T_e.

The time sequence type ($TS[T]$) of TOODM is based on the same concept previously defined by Segev and Shoshani [80]. In [80] temporal characteristics of data and operators over them are defined, especially for time sequences. This is adopted and incorporated into TOODM as the basic notions of the time model.

The TOODM is by no means fully documented in [69], however it claims to support time-varying type specifications as well as time-varying data. In the former case specifications of both attributes (or instance variables as it is called), messages/methods, and constraints are allowed to change over time. This is handled by defining user-defined types as subtypes of class V-Class. (Time-varying data is explained above.) This means that TOODM supports both type evolution/versioning[22], and data histories and predictions.

3.5.3 Models Characteristics and Summary

A main difference between the two models presented above is that $T_$Chimera has a more formal definition, but neither of the two has a strict formal definition. Table 3.2 gives a brief overview of the characteristics of the $T_$Chimera and TOODM models. In more technical terms TOODM supports, or is at least believed to support, a broader range of model and language characteristics, such as multiple time dimensions, possible branching future times, type versioning, a rudimentary constraint language, and a query language. Although these characteristics are described they are not well defined in the same manner as the characteristics of $T_$Chimera, that seems more robust and consistent throughout. Temporal query languages for TOODM are also developed, three related languages exist; the TOSQL [69] was the first version of the SQL-like language for TOODM, TOOA [70] is the temporal algebra defined, and at last TOOSQL [71] (based on TOOA).

Other Temporal Extended Object Models In this class of models there exist other proposals which incorporate time as an integral part of the model. First, Clifford and Croker [22]

investigated some intuitively given temporal aspects and properties that should be supported with the notion of objects[23]. Their focus is on the concept of *historical* objects, i.e. instances of ADTs, where an object identity plays the central role. Their motivation is based on the fact that both for Entity-Relationship and relational models user-supplied keys are often

[22]Type evolution/versioning is not to be interpreted as schema versioning. But, it is an analogue to it, in the sense that schema version's validness is time dependent, and timestamped.

[23]Here this means first-class objects, i.e. with a system defined OID.

Model Charact.	***T*_Chimera**	**TOODM**
Time Structure	linear & discrete	branching(?) & discrete
Time Dimension	valid	valid, event & trans.[1]
1.class objects	objects	objects collections
Temp. attr. value	formally as an instant-function pair	an element of a time sequence
Versioning	attributes	attributes types
Lifespan	contiguous	non-contiguous[2]
Intent & Extent	type & class	type & class[3]
Inclusion & Substitute.	yes	no
Query Language	no temp. QL	TOSQL, TOOA & TOOSQL

[1] Transaction time always supported, user-defined dimensions may be added
[2] Applies to intervals, meaning that consecutive intervals need not be adjacent
[3] Not a clear type-class dichotomy.

Table 3.2: A brief summary of T_Chimera and TOODM

time-varying, and, therefore, do not provide unique identifications of objects over time. The (temporal) features of this rudimentary model are; it supports valid time, attributes are interval timestamped, functions are first class objects with function types, temporal attribute values are encapsulated as functions, object's lifespan is non-contiguous (i.e. an object may be logically deleted and subsequently reincarnated). However, they do not discuss issues related to inclusion semantics and substitutability. In essence this model seems to be an attempt to make its relational counterpart of N1NF (non-first normal form) relational models object based. See [21, 40, 92] for examples of N1NF relational models. [21, 40] also define attributes values as functions similar to the object model above.

Su and Chen's OSAM*/T

model [91] is also an approach of this class, however it is a model that is designed for knowledge representation. Except for mechanisms for handling knowledge rules, such systems do not necessary advocate totally different approaches compared to a more general object data models. OSAM*/T is the temporal extension of the knowledge representation model OSAM* (Object-Oriented Semantic Association Model). The most interesting feature of OSAM*/T is that it limits the notions of time. The only support of time notions in OSAM*/T are a Start-time and an End-Time. For ex-

ample the time dimension(s) for the management of temporal information as valid time or event time is strictly defined by the application. Because temporality affects only some data and not necessary other data of the same type, Su and Chen argue that there is no need to explicitly declare the whole type (and thereby all objects of the type extent) as temporal. For example, when Mary gets a raise in salary this only affects the corresponding object instance named 'Mary' in the database. Instead of having a timestamped salary attribute associated with 'Mary' and all other instances of the type, the exact semantics of the temporal information (i.e. valid and/or event times) is expressed by means of knowledge rule(s), which is (are) data in its (their) own right. Rules are often associated with single object instances[24]. In OSAM*/T rules are also used to handle what is best compared to attribute versioning of other models. On the other hand the transaction time notion of OSAM*/T is supported through an object timestamping mechanism. A timestamp is defined for object instances only, and the object history, that manages which classes that an object has appeared in and appears in, is derived form the individual object instance histories of that object. In a similar manner OSAM*/T supports association histories. Because object classes are defined by which associations they have with other classes and that association types are them-self defined as classes, the association histories could also be derived by means of object instance histories. The object timestamping approach is used to separate both logical and physical current data from historical data. OSAM*/T has also its own query language, OQL/T, which is an temporal extension of the OQL [4] used in OSAM*.

3.6 Temporal Application Tailored Models

In this section we present some models which have been developed to meet requirements of specific application domains. The earliest developments with temporal databases extended the relational model and system. However, some applications have needs and requirements towards a database system that are not met by any relational system. Not surprisingly these application are typical for disciplines such as design, medicine, software engineering, office automation, statistics, scientific experiments, geography, cartography, etc. Databases of these disciplines require features to handle complex structures, version and configuration management, multiple data types, and/or different views, scales and resolutions of data. In principle, object databases are recognized to be better suited for these purposes compared with relational databases.

Based on these specific requirements several temporal object models have been developed either to model temporal extensions to existing and specific

[24] An object in OSAM*/T can appear in several classes simultaneously, and the notion of an *object instance* is used to denote an object's appearance in a particular class.

features, or using time as a means to solve the problem of the application required features. Even though, these proposals are specific, because of their contexts or features, they do indeed have general interest. We could divide these proposals into two main subclasses as mentioned above; first, one that extends existing features with time, and the other that uses time as a means to solve specific needs directly. We do not want to do so, because we feel that it will be much of a "chicken and egg" classification. Instead, a better approach is to find some common denominators and focus on these, and give examples from the proposed models found in the literature. We then have the following subclasses: versioning and change management, time-series management, and multi-media handling.

Although we could add others subclasses as well, such as real-time, distributed, active and deductive databases, and object management, we stick to those subclasses mentioned. In the following the former class,versioning and change management, is presented into some detail. The latter two classes, time-series management and multi-media handling, are rather briefly presented so that the reader gets acquainted with the strength and potential of time support in two modern, still limited, areas of advanced database management systems.

3.6.1 Versioning and Change Management

A vast set of proposals fall into this class [19, 52, 51, 54, 56, 72, 89, 55, 77]. That is, the proposals' main objectives address the needs for version and configuration management, and/or schema evolution and versioning.

3.6.1.1 Object Versioning

Object versioning, in the context of databases, provides models and mechanisms for management of different versions of objects, and how these versions are configured and/or related, i.e. how they are associated with each other into more complex objects. Versions of an object could be defined to represent histories (better known as revisions or change histories[25]), alternatives (different properties), variants (distinguishable through parameter/property values), etc. Katz has given a comprehensive presentation of these feature in the context of engineering databases [54].

The version types listed above do not necessarily involve time directly. The only type which directly reflects time is the history type. For example, a set of revisions makes up a set of consecutive, ordered states of an object, e.g. of a design object. That is, the set is ordered either by a successor relation which has an implicit reference to time (before/after), or each revision is timestamped giving an explicit time ordering relation. A system

[25] A revision or a change history is in this context analogue to temporal data presented in the previous sections.

can provide both techniques because they only represent different views of the same notion. Käfer and Schöning strictly distinguish in [52] between the concepts of object versioning and that of object histories. They argue that versioning is something that reflects the nature of an application and is handled by the user. Object versioning implies management of complex objects which contains branches and alternatives, as they have defined for the MAD model [51]. Object histories on the other hand are the developments of objects keeping track of the changes made to it[26], independent of it being an other type of version as well. This part is presented in the temporal extension to the MAD model, TMAD [52]. Hence, an object version could have its own history. That is, a history of one particular object version may be regarded as a subset of the object history of the entire object. An object history of the entire object is in that sense implicitly given by the partial ordering of its object versions.

Object histories are in the context of Käfer and Schöning the primary concern of temporal databases which handles timestamps. Still, both object versioning and object histories manage temporal data. This will be clear in the following. For example, an engineering (or a design) database has to supply both the notion of a version and the notion of a history [52]. The versioning concept as presented both by Katz [54], and by Käfer and Schöning [51], is related to the transaction time dimension. Moreover, several object versions of the same parent object can be current in the database at the same time, i.e. simultaneously. They do not supersede each other, rather, they represent parallels of the same object. This extends the notion of transaction time as a linear ordering of database states (or events), to a notion where transaction time is branching and the database is managing multiple, parallel states of the same objects. The version model by Chou and Kim [19, 55] defines object versions (i.e. siblings) to be in *version-of* relationships with their generic object (i.e. parent) to which they belong, and object versions are in *derived-from* relationships among versions of the same generic object. The MAD model by Käfer and Schöning [51] also shares the distinction between a generic object and an object version. A *derived-from* relationship defines which version is derived from an other version. For example, each version in Chou and Kim's model has a transaction timestamp. This derivation hierarchy is a directed acyclic graph of versions (a version may be derived from one or more existing versions), and is a partial order of versions. This structure also represents the transaction time as a partial order because each version is a function of a corresponding transaction that created it in the database. Versions in a derivation hierarchy represent discrete points, and therefore it's natural to look at versioning as a type of (implicit) temporal data management. Revisions and change histories of

[26] Object histories here, most presumingly, representing valid-times. Even though, they don't explicitly state that it is valid time.

objects are orthogonal to these graph structures.

The VISION model by Sciore [76] applies an other approach. (This model is an extension and revision of the VISION model by Caruso and Sciore [13].) The model uses annotations which manage the different aspects of versioning, and it was an attempt to define a uniform treatment of versions and temporal data. Three types of versions are defined. Historical versions are the same as valid time data in a temporal database. Revised versions are the same as revisions mentioned above, and correspond to transaction time data in a temporal database. Finally, alternatives are analogous to an object version graph. The three types of annotations are introduced by three predefined annotation classes, i.e. *HistoryFn*, *RevisionFn*, and *AlternativeFn*, respectively. Each one of these classes have their own set of properties and methods. For example *AlternativeFn* has method *newAlternativeFn* for defining new alternative versions. VISION supports only attribute versioning and therefore does not have a strict datamodel distinction between a generic type and a version type. For example, say a salary history attribute of an `employee` type is defined as `salary-h: HistoryFn(RevisionFn(salary))`. Here salary is, in VISION terms, an historical attribute represented as a set of $(date, value)$ pairs. But, for each such pair also a set of revisions exist making up the database history of the `salary-h` attribute. Any combination of these three classes may be used to define versions, e.g. if we enter the terms *RevisionFn* and *HistoryFn* in the opposite order, then the `salary-h` defines the attribute as a collection of database updates with each update having a collection of valid-time data. Though, the semantics of these definitions remain the same.

The version types listed at the beginning of this section are all orthogonal concepts. For example, the time dependent type, revision, can be used to model revisions of variants, alternatives, and even of configurations. Thus, all of the other version types may be defined to have revisions as well. Beware, in fact, that the above notions of transaction time introduce a differentiation between what may be called (timestamped) database objects and (timestamped) user-defined objects. That is, referring to the differentiating made by Käfer and Schöning, an object version if something different from that of a revised version in that the former may be identified by a version number or name, and the latter is solely identified by its timestamp. Furthermore, object versioning typically define a partial ordering of versions whereas at least revisions, based on timestamping alone, define a total ordering. This is why object version graphs do exhibit an additional notion of predecessor/successor, e.g. a *derived-from* relationship, to handle the partial ordering. One may argue that an object version graph, because it is user-defined and -controlled, is better characterized as reflecting the notion of valid time. It possibly may, but the semantics of transaction time still holds because the meaning of being a version is based on the fact that a version when first created does not change. If it changes its attributes it, then, only gets a new revision. Its version identification should remain the same, and

will always reflect a particular node in the graph. The semantics of valid time do not fit this scenario because valid time may promote both retro- as well as pro-active changes.

3.6.1.2 Schema Versioning

So far, the notions of both versions and revisions have only concerned the database extent, i.e. data instances. Schema evolution is an other area of concern, i.e. revisions to a schema gives several consecutive versions of that schema. Schema evolution in traditional databases usually denotes a situation where a new or changed schema overwrites the schema in operation, possibly, because of new needs of the application. See for example ORION [56] for this approach. By means of a temporal database (with explicit or implicit reference to time) older data and also older schemata need not to be discarded. In such a context we define schema versioning to be a technique that support schema evolution – the revision history is maintained. The difference between the old type of schema evolution and that of temporal databases is that the former deals only with one schema, whereas the latter logically deals with multiple schemata[27].

Schema evolution is associated with transaction time, and, hence only one schema version is in effect at any referenced point in time. That is, every new schema version only supersedes its previous versions at the point in time it comes into effect. However, if we query the database with a reference time (e.g. what was Mary's salary as best known at time t) or rollback the database to a previous state the schema version in effect at that particular time or state will be used. Some object models support schema versioning, such as MATISSE [1], and POSTGRES [72, 89]. A schema may be regarded as an object and each new version of the schema is a revision of the previous schema version, and a total order is imposed by transaction times. For instance, MATISSE does this by explicitly timestamping each schema version.

The above schema evolution strategy is based on a total order of versions. A proposed model by Kim and Chou [55] support an other notion of schema versioning. That is, each new version is created from a logical schema (the main or parent schema). Such a version could for example represent a user's view of the data. This notion of schema versioning is orthogonal to that of schema evolution above, and could be regarded as an analogue to *alternatives* mentioned for object versioning. Then, if both schema evolution and the notion of schema versioning as defined by Kim and Chou [55] are supported, the schema change management system supports a branching transaction time. That is, each schema version may have its own schema evolution (i.e a

[27] Historically there has been a clear distinction between schema versioning and schema evolution, but in the context of temporal databases this distinction seems rather odd.

set of revisions) without each time having to create a new version from the main or parent logical schema.

Besides the transaction-time schema versioning also valid-time based schema versioning is possible (see below). A valid-time schema versioning is defined to support both retro-active and pro-active schema changes, which is not possible based solely on transaction time versioning as described above. An other effect of valid-time versioning is that a change may span several transaction ordered versions. Thus, queries with reference time *now* (i.e. as best known now) may access historic data, but the schema version in effect has all known retro-active updates included. Queries with a *past* reference time (i.e. as best known then), similar to the transaction time-oriented schema versioning approach, have access to previous versions of the schema as they were at the referenced time. An open question remains, though; does a single notion of reference time (as in the example above) apply to both intentional and extensional data in a setting where bi-temporal schema versioning is supported? For example, is the following a meaningful query: "*find all employees who worked on projects during 1995, as best known now, but as specified at that time.*"? That is, are the reference times, '*as best known now*' and '*as specified at that time*', applicable both to the data extent and the data intent? It is a plausible situation that the former of the reference times refers to the database extent, and the latter refers to the database intent. Then, this query will use the most up to date data, but possibly it refers a previous schema with a data structure that is not necessary equivalent to the structure the actual data is stored under. How such relationships between intent and extent reference times should be coped with is a future research issue. However, Clifford and Isakowitz [28] have studied a single notion of reference time. That is, informally, a reference time is to view a database at a particular state. We believe, however, that there are situations where reference time should distinguish between the intent and the extent of a database, and the interaction between time dimensions may vary from query to query of extent and intent reference times. In the case of Clifford and Isakowitz's reference time, these issues were never raised because they only considered the extent of a database.

The model by De Castro et al. [33] defines a bi-temporal schema versioning for the relational model, and is, to our knowledge, the only explicit example supporting valid-time schema versioning. No object models have explicitly defined a valid-time schema versioning approach. TOODM is with its type versioning model [69] close to this, but the model does not explicitly define what is meant and what is the consequences of the (bi-)temporal type definitions besides stating that object properties definitions may change over time. Moreover, schema versioning in TOODM is not related to any known schema evolution and versioning strategies.

3.6.1.3 Summary of Object and Schema versioning

Table 3.3 gives a brief overview of some of the models which support versioning. The first column lists the Models. The second column names (and indicates) the type of object versioning supported. The third column indicates how each model incorporates an explicit or implicit temporal dimension. The fourth and fifth columns do the same for schema versioning as columns two and three do for object versioning. Entries with 'n/a' and '–' mean not applicable and not relevant, respectively. Entries having a '(?)' indicate that the version mechanism or model is not well defined in the literature referenced.

Name	Object Versioning by	Temporal Approach	Schema Versioning by	Temporal Approach
MATISSE	object graph	timestamp & version ID	schema is an object	timestamp linear
POSTGRES	object graph	timestamp	time-varying catalogues(?)	timestamp linear
ORION	version-derivation hierarchy	version ID	schema evolution[1]	none
TOODM	n/a	–	type versioning	bi-temp. timestamps
T-OODAPLEX	object graph	arbitrary	n/a	–
(T)MAD	version-derivation graph	derived-from relationsh.	n/a	–
VISION	annotation	naming & priorities	n/a	–
TIGUKAT	version slices(?)	bi-temp. timestamps(?)	version slices(?)	bi-temp. timestamps(?)
EXTRA-V	attributes	multi-dimensional attributes	n/a	–

[1] The new schema overwrites the old schema

Table 3.3: Object and Schema Versioning Approaches

Only TOODM and (possibly) TIGUKAT of the models presented in Table 3.3 support some aspects of valid-time schema versioning. We see that some of the previously presented proposals also support various kinds of change management. All the above mentioned models do support valid-time and/or transaction time for handling historical (and predictive) data and revisions in temporal databases, respectively.

Traditionally, historical versions, revised versions, and object versioning (as in Table 3.3) are regarded as different concepts. However, Sciore tries by his EXTRA-V model [77] to unify these concepts on a conceptual level[28].

[28] The results presented with the EXTRA-V model is based on the VISION model [76,

He refers to three areas where versioning has been a prominent database research issue, namely within CAD, CASE, and temporal databases (called historical databases by Sciore). He then argue that each area has taken different approaches often tailored by different needs:

Temporal databases and versioning, as presented in Section 3.4 and Section 3.5, deal with data as a function of time, recording the changes in the real world and presumingly when these changes are reflected in the database.

Database research within CAD (Computer Aided Design) also deals with versions (revisions and alternatives). The focus has been on general models and having a system level understanding of versioning. Therefore, the aspects of version management and mechanisms for long transactions have had special attention. Less attention has been put on aspects of data models and declarative access to versions.

The main emphasis on database research within CASE (Computer Aided Software Engineering) is put on version configuration of software modules. The main application has been software maintenance, such as managing bug-fixes and new releases.

These three areas of research have been, according to Sciore [77], focusing on versioning on different abstraction levels and/or tailored by different application needs.

The main objective of EXTRA-V is to conceptually capture the semantics of that an entity is being versioned. Sciore claims that this conceptual information is lost by the *generic* type and *version* type models, which introduce associations among generic objects and their versions. And, hence, that an entity is versioned is conceptually handled by the application. For example, notions and operations like *insert/new, update*, and *delete* versions are not part of the model and its language(s).

The approach taken by EXTRA-V is to distinguish an object's versioned attributes and the non-versioned attributes. All versioned attributes of an object type are grouped under a keyword *versioned*. They represent the versioning of an object. Every versioned attribute can be multi-dimensional and representing a multi-dimensional value. The versioning dimensions are arbitrary and user defined. Any combinations of temporal and versioning dimensions are possible. The EXTRA-V model seems to unify several of the features found in the three areas of version management.

3.6.2 Time Series Management

Time series is an important data type in many scientific- and/or statistical-based applications, but also in a vast set of other domains. This has caused a growing interest for explicit time series models in data management and databases, typically deployed in economics, finance, assurance, experimental

75]. The EXTRA-V is an versioning extension of the EXTRA model by Carey et al. [12].

and empirical sciences.

This section presents principles and definitions concerning time series, as well as proposed models for time series support in databases.

3.6.2.1 Principles and Definitions

Basically, a time series is a data types which defines a set of $(time, value)$ pairs[29], where an order may be imposed by the time parameter. However, a time series could be more specialized and captures for example flavor of periodicity, cyclic behavior or properties of a phenomenon, and infinite (i.e. ever growing) sequences. Even different time granularities of time series come into consideration when different time series are integrated or joined, and/or represent a multi-variate nature.

Tuzhilin and Clifford [95] treat periodicity (related to events) in three different ways, meaning that, if a time series is periodic[30], then events occur either as periods or repeated cycles, at regular intervals, or now and then; intermittent.

In the first case events in a time series occur equidistantly, i.e. any two adjacent events are separated by the same distant in time. For example, a traffic light shifts from red to green after one minute, and from green to red the next minute, and so forth.

In the second case events in a time series occur within an interval, i.e. any two adjacent events are separated by a minimum and up to a maximum distant in time. For example, a traffic light shifts regularly between every thirty second and every second minute during a day. That is, during office hours it shifts every minute, during rush hours it shifts every second minute, and the rest of the day it shifts every thirty second.

In the last case events in a time series occur "intermittently", i.e. any two adjacent events are separated by an unknown distant in time. For example, any traffic light breaks down or does not function correctly from time to time, but when such unfortunate situations occur are unknown in advance, still, situations of this kind are believed to occur sometime in the future. In a sense we may say that such situations occur periodically (e.g. on average five times a year), but initially they are considered to be irregular events, i.e., we don't know when to expect them, only that they will occur.

Tuzhilin and Clifford have formalised the first two notions of periodic data [95], which they coined *strong periodicity* and *near periodicity*, respectively. In the same paper they also showed various strategies for adding periodicity to the TSQL2 standard [88].

[29] The parameter *value* could be of a simple literal type, a structured literal type, or an object type.

[30] Tuzhilin and Clifford used the definition of 'periodic' found in American Heritage Dictionary [6].

A calendar is the base for the unit of periodicity. There are different calendars, such as business calendars (with 5 days per week), Gregorian, or Islamic, etc. A calendar capture the notions of both periodicity and infinite sequences. For example, a new week begins after every seventh day, and the number of years is infinitely countable, respectively. On the other hand non-periodic time series could be represented by an ordinal calendar — being isomorphic to the natural numbers [74].

Managing temporal data with various granularity is a data integration problem that is not only restricted to integrating data from different data sources. Even data representing the same real world entity may have different time granularities among its defined properties. Wang et al. [96] discuss and formalize these issues in context of a temporal relational model by means of defining new notions of temporal functional dependencies and temporal normalization. They also illustrate the problem with an example of an ACCOUNT entity type which specifies transactions, balance, accumulated interest, and time information of each account. The temporal issue arise because transactions and balance values are recorded with a granularity of seconds, whereas accumulated interest values are recorded or updated only every twenty-four hour. Even though object databases are not normalized in the same sense as that of relational databases similar issues concerning treatment of multivariate time series in object databases should be stressed. ACCOUNT data could for instance easily be modeled as an object type, capturing multivariate time series.

3.6.2.2 Time Series Models and Systems

A few time series models have been proposed [36, 34, 74, 78, 81]. There also exist some commercial and non-commercial systems that explicitly manage time series, such as Illustra [47] and FAME [38] of the first category, and of the second category we have CALANDA [35] and a POSTGRES-based prototype [17].

We could characterize the above proposed models and systems into two main categories; one category representing pure time series management systems (TSSM). The other category represents time series as an extended or extensible feature of more general purpose temporal data management systems.

Of the first category we briefly present the CALANDA system [35] and model [36, 34, 74] developed by a group at Union Bank of Switzerland. The non-object based system FAME [38], by FAME Software Corporation, also falls into this category.

The idea behind the CALANDA model and system is based on the experience that temporal data models (and databases) do not capture the inherited requirements of time series management, including both structural

and functional aspects. This is most prominently stated in [74][31], where they give a comparison with both FAME and systems of the other category.

CALANDA is an object-oriented system, defined as a special purpose time series management. Its main features and modeling constructs are some predefined classes for modeling times series and groups of time series. A time series could be of a simple or multi-variate event type, i.e. an event may record one or multiple attributes, respectively. An attribute are either simple valued or of an array type. A group is an aggregation of time series of the same type or of different types. All time series data, groups included, may define header data which is common to the whole time series. The time handling, seen from the users point of view, is based on calendars and operations on calendars. They stress that CALANDA is tailored towards database capabilities, but CALANDA also offers a manipulation language that is said to be computational complete. Thus, an application required functionality could, in principle, be tightly integrated with a CALANDA database by a database designer.

The other category of time series includes models defined by Segev and Chandra [78], and by Segev and Shoshani [81], both based on the notions of time sequences found in the TOODM model presented in Section 3.5.2. Systems include the above mentioned POSTGRES-based prototype [17] and Illustra [47]. We briefly present the Illustra system.

Illustra time series is based on a predefined *DataBlade module* [32], which treats time series as first class objects known to the system. This module includes specific data types and constructors for managing time series (only regular time series at the moment), including calendars, and operations (analysis, aggregations, updating, etc.). One of the advantages of Illustra, compared for instance to CALANDA and because of its general purpose nature, is that time series could easily be integrated with other non-time series DataBlade data.

3.6.3 Multi-Media Handling

Multi-media data models extends the set of basic data types we are acquainted with in databases and programming languages, such as integer, float, character, and boolean. Typically a multi-media data model is characterized by basic data types such as text, image, graphic, animation, video, and audio. Some relational database systems have to some degree incorporated primitive and simple structures for multi-media data handling, better

[31] They argue that time series has been "a neglected issue in temporal database research", based on experiences gained in domains such as finance and banking. Hence, it could be argued that, at least some of, the requirements encountered could be restricted to those domains.

[32] In general DataBlade modules in Illustra are user extensions to capture requirements of specialized domains.

known as binary large objects, BLOBs[33]. That is, such data types are defined as an attribute of a relation, like the approach taken by for example Sybase and Oracle.

In an object-oriented setting multi-media objects are characterized by defining properties and operations that handle such objects. The idea is to either support such types in a similar manner as the support of other basic data types, or as intrinsic object types, where instances represents first class objects in their own right.

The needs for multi-media data support are found in most modern organizations and businesses. These application include design systems, publishing and media systems, office-automation, medical record and image systems, tourist information kiosks, and so forth.

3.6.3.1 Temporal Support

The temporal support for multi-media databases covers two aspects of management of such data. One captures the temporal structural aspects, such as histories, revisions, and versioning as presented in the previous sections. The former approach have been presented by for example Adiba [2], Adiba and Quang [3], and by Chu et al. [20].

The other aspect of time and multi-media is specific to continuous multimedia types, such as audio, video, and animation. For example, it is important to synchronize audio and video both as database output, and to define such relationships among the database objects. Related to synchronization is the presentation of a certain amount of output by a certain speed that is defined by a temporal *quality of service*, QoS [41, 65]. The notions of QoS in context of databases have been accentuated because of multi-media data. In a similar way the introduction of multi-media systems also changed the notions of a document, and does not represent a static kind of entity. Thus, (parts of) continuous data are integrated into or with the document concept, and such data may be constrained according to temporal relationships [37]. For example, a video stream could be selected by a temporal partition of another video source, or an (transparent) image could act as an overlay to a video sequence or animation for a defined set of time instants.

3.6.3.2 A Video Database

The proposal by Hjelsvold et al. [45, 46] deal with an temporal foundation of, and searching and browsing shared video databases. The interesting features of this approach is that it is structuring video data based on the model of *film theory*. Basically, in such a model a film is constructed or combined by several film pieces, where the different combinations define different *contexts*, and

[33]The semantics of a BLOB, i.e. what multi-media data type it contains, the internal bit structure etc, are most often interpreted by the application.

where each context describe a story or part of a story, giving it a meaning. Their application is a video archive for handling television news program, e.g. enabling producers and journalists an easy access to video pieces (e.g. through querying or browsing the video archive), use of video pieces in new contexts, and store new contexts as an object relating existing and/or new video pieces.

The structure of the generalized archive covers both logical and physical aspects of video data. At the most abstract level is the *video document*, that is a logical *video stream*. Each document consists of components, *video context indexes* or *annotations*, that define the relationship between the real world "story" and its video data. An annotation identifies *stream intervals* of contiguous sequences of *media streams*. A media stream is a generalization of *stored media segments* or video streams. A stored media segment is the physical piece of data stored and generated due to video recording. The stream structure of video could easily be generalized to audio and, possibly, other continuous media.

An integration of (simultaneous) different continuous media streams could easily be achieved by specializing the notion of annotation, for the purpose of, for example, combining audio and video that originally are not based on the same recording. By means of this structure an specialized annotation would logically define, and possibly encapsulate, Allen-like temporal relationships (e.g. during, overlaps, before, etc. [5]) between a video stream and an audio stream, but they would represent different media segments. However, such structures are not explicitly defined for annotations in the present model [45, 46].

The above presented video database does not directly introduce the notions of valid- and transaction-times as defined by temporal databases. In contrast, Eini's work does exactly that, where both valid- and transaction-time dimensions, as well as a novel *play-time* dimension, are introduced [37]. The former two are used in the conventional sense, but, now also in managing multi-media data types. The latter, play-time, is defined to sequentially relate units of media data to each other in a temporal order. For example, video frame `i` comes before video frame `i+1`, and audio sample `j` comes before audio sample j+1. The play-time dimension is discrete, and the smallest unit of time of one particular defined play-time dimension is dependent on the data type for which it is defined. For example, the Logical Data Unit (LDU), a multi-media concept, denotes the smallest unit of data managed, and the PAL standard recommend that video is played with 25 frames per second. That is, a duration of a video LDU is $\frac{1}{25}$ of a second and the *chronon* of a play-time dimension for a PAL video is, therefore, its *LDU duration*.

Chapter 4

Summary

The previous sections surveyed three major approaches to incorporate time into an object model environment. The three approaches are, informally, classified into a temporal extensible object model, a temporal extended object model, and an application-oriented temporal object model, respectively.

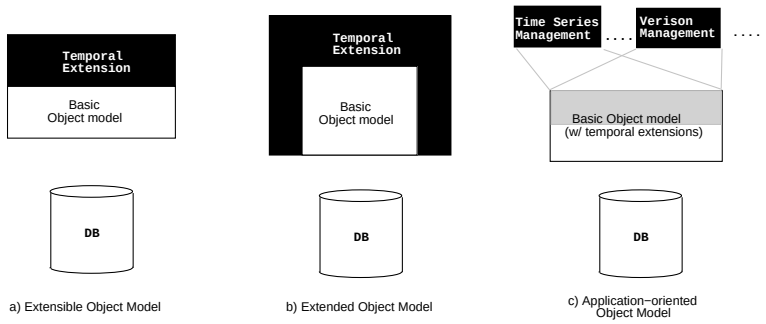

Figure 4.1: The three classes of temporal object models

This classification is illustrated in Figure 4.1, and also shows how it may have an impact on the database system architectures. In Figure 4.1 a), which is typically represented by T-OODAPLEX, no database system internal changes are made. The temporal object model is given by an additional layer on top of the basic object model. Hence, data are treated internally as non-temporal data only.

The second class extends the notions of a basic object model with temporal capabilities. That is, it introduces new definitions, concepts, and language constructs that are an integral part of the model itself. It will also influence the system internal manipulation and access of data in the database. This is shown in Figure 4.1 b).

The last class is defined to meet specific needs of different application domains, such as time series management in statistical and scientific databases,

and version/configuration management in design databases. Although these systems may have extended the model kernel with notions of time series and versions, respectively, the underlying basic model and its eventually temporal extensions may in principle be based on either of the former two classes. This is illustrated in Figure 4.1 c).

Even though the three classes impose differences, such as database internal awareness of temporal semantics (e.g. constraints, value domains), temporal query language support, emphasis on temporal support, etcetera, they also share several similar principles. The most predominant are those provided by the structural and behavioral aspects of the object-oriented paradigm. Hence, object-orientation enables a designer to extend a database model to nearly include whatever structure and behavior required. In the (temporal) relational world extensions to a model has to be realized as part of an application. In object-oriented systems, missing features could be defined by structural and behavioral extension by means such as user defined object types. Extensibility is an inherited property of the object-oriented paradigm. In that respect an object-oriented system is more flexible and capture implicitly the modeling feature of all conceptual temporal requirements.

Bibliography

[1] ADB. *MATISSE 2.3 Technical Overview – Release 2.0*, 1995. http://www.adb.com.

[2] M. Adiba. STORM:A Structural and Temporal Object-Oriented Multimedia Database System. In *International Workshop on Multi-Media Database Management Systems (IW-MMDBMS)*, Blue Mountain Lake, NY, August 1995.

[3] M. E. Adiba and N. Bui Quang. Historical Multi-media Databases. In Y. Kambayashi, editor, *Proceedings of the International Conference on Very Large Data Bases*, pages 63–70, Kyoto, Japan, August 1986.

[4] A. M. Alashqur, S. Y. W. Su, and H. H.Lam. OQL: a query language for manipulating object-oriented databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 433–442, Amsterdam, Holland, 1989.

[5] J .F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[6] American Heritage Dictionary. Houghton Mifflin Company, second edition, 1985.

[7] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pages 223–240, Kyoto, Japan, December 1989.

[8] J. van Benthem. *The Logic of Time*. Kluwer Academic Publishers, Dordrecht, Bosten, London, 2nd. edition, 1991.

[9] E. Bertino, E. Ferrari, and G. Guerrini. A Formal Temporal Object-Oriented Data Model. Technical Report 141-95, University of Milano, 1995.

[10] E. Bertino, E. Ferrari, and G. Guerrini. A Formal Temporal Object-Oriented Data Model. In *Proceedings of the International Conference*

on Extending Database Technology, volume 1057 of *Lecture Notes in Computer Science*, pages 342–356. Spinger-Verlag, 1996.

[11] M. Böhlen, C. S. Jensen, and B. Skjellaug. Spatio-Temporal DataBase Support for Legacy Applications. Submitted for publication, April 1997.

[12] M. J. Carey, D. J. DeWitt, and S. L. Vandenburg. A data model and query language for EXODUS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 413–423, Chicago, IL, June 1988. ACM.

[13] M. Caruso and E. Sciore. Meta-Functions and Contexts in an Object-Oriented Database Language. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 56–65, Chicago, IL, June 1988.

[14] M. Caruso and E. Sciore. *The Vision Object-Oriented Database Management System. In Advances in Database Programming Languages*, chapter 9, pages 147–163. ACM Press, New York, NY, 1990.

[15] R. Cattell, editor. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, San Mateo, 1994.

[16] R. G. G. Cattell. *Object Data Management.* Addison-Wesley, revised edition, 1994.

[17] R. Chandra and A. Segev. Managing temporal financial data in an extensible database. In *Proceedings of the International Conference on Very Large Data Bases*, pages 302–313, Dublin, Ireland, September 1993.

[18] T. S. Cheng and S. K. Gadia. An Object-Oriented Model for Temporal Databases. In R. T. Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, pages N 1–19, Arlington, TX, June 1993.

[19] H.-T. Chou and W. Kim. A Unifying Framework for Version Control in a CAD Environment. In *Proceedings of the International Conference on Very Large Data Bases*, pages 336–344, Kyoto, August 1986.

[20] W. W. Chu, I. T. Ieong, R. K. Taira, and C. M. Breant. A Temporal Evolutionary Object-Oriented Data Model and Its Query Language for Medical Image Management. In *Proceedings of the International Conference on Very Large Data Bases*, August 1992.

[21] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of the International Conference on Data Engineering*, pages 528–537, Los Angeles, CA, February 1987. IEEE Computer Society Press.

[22] J. Clifford and A. Croker. Objects in Time. *IEEE Data Engineering*, 7(4):189–196, December 1988.

[23] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, 19(1):64–116, March 1994.

[24] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "Now" in Databases. *ACM Transactions on Database Systems*, 1997. Tentatively scheduled for Vol. 22, No. 1, March 1997.

[25] J. Clifford, C. Dyreson, T. Isakowitz, S. J. Jensen, and R. T. Snodgrass. *"Now"*, chapter 20. In R. T. Snodgrass (ed.) [88], 1995.

[26] J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In S. Navathe, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 247–265, Austin, TX, May 1985.

[27] J. Clifford and A. Tuzhlin, editors. *Recent Advances in Temporal Databases: Proceedings of the International Workshop on Temporal Databases*, Workshops in Computing, Zurich, Switzerland, September 1995. Spinger-Verlag.

[28] James Clifford and Tomas Isakowitz. On the Semantics of (Bi)Temporal Variable Databases. In *Proceedings of the International Conference on Extending Database Technology*, volume 779 of *Lecture Notes in Computer Science*, pages 215–230. Springer-Verlag, 1994.

[29] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[30] E.F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems*, Courant Computer Science Symposium 6, pages 65–98. Prentice-Hall, 1972.

[31] O.-J. Dahl, B. Myrhaug, and K. Nygaard. (Simula67) Common Base Language. Technical Report N. S-22, Norwegian Computing Center, Oslo, Norway, October 1970.

[32] U. Dayal. Queries and Views in an Object-Oriented Data Model. In *Proceedings of the 2nd. Workshop on Database Programming Languages*, Oregon, USA, June 1989.

[33] C. De Castro, F. Grandi, and M. R. Scalas. On Schema Versioning in Temporal Databases. In Clifford and Tuzhlin [27], pages 272–291.

[34] W. Dreyer, A. K. Dittrich, and D. Schmidt. An Object-Oriented Data Model for a Time Series Management System. The CALANDA Time Series Management System, UBILAB, Union Bank of Switzerland, Zurich, Switzerland, November 1995.

[35] W. Dreyer, A. K. Dittrich, and D. Schmidt. Using the CALANDA Time Series Management Series. The CALANDA Time Series Management System, UBILAB, Union Bank of Switzerland, Zurich, Switzerland, nov 1995.

[36] Werner Dreyer, Angelika Kotz Dittrich, and Duri Schmidt. Research Perspectives for Time Series Management Systems. *ACM SIGMOD Records*, 23(1):10–15, March 1994.

[37] I. Eini. A Temporal Object-Oriented Multimedia Data Model (preliminary title). Master's thesis, Department of Informatics, University of Oslo, 1997. Forthcoming.

[38] FAME Software Corporation. *User's Guide to FAME*, 1990.

[39] S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[40] S. K. Gadia and C. S. Yeung. A Generalized Model for a Relational Temporal Database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 251–259, Chicago, IL, June 1988.

[41] V. Goebel and T. Plagemann. Data Management and QoS in Distributed Multimedia Systems – Towards an Integrated Framework. In *Proceedings of the Fourth International IFIP Workshop on Quality of Service*, pages 79–83, Paris, France, March 1996.

[42] Adele Goldberg and David Robson. *Smalltalk 80: the Language and its Implementation.* Addison-Wesley, Reading, Mass., May 1983.

[43] I. Goralwalla and M. T. Özsu. Temporal extensions to a uniform behavioral object model. In *Proceedings of the International Conference on Entity-Relationship Approach*, Dallas, TX, June 1993.

[44] G. Guerrini, E. Bertino, and R. Bal. A Formal Definition of the Chimera Object-Oriented Model. Technical Report IDEA.DE.2P.001.01, IDEA – ESPRIT Project 6333, 1994.

[45] R. Hjelsvold, R. Midtstraum, and O. Sandstå. A Temporal Foundation of Video Databases. In Clifford and Tuzhlin [27], pages 295–314.

[46] Rune Hjelsvold, Roger Midtstraum, and Olav Sandstå. Searching and Browsing a Shared Video Database. In *Proceedings of the 1995 International Workshop on Multi-Media Database Management Systems*, pages 90–98, Blue Mountain Lake, New York, August 1995. IEEE Computer Society Press.

[47] Illustra Time Series Support. White Paper, Informix Software Inc., 1996.

[48] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia [eds]. A Glossary of Temporal Database Concepts. *ACM SIGMOD Records*, 23(1):52–64, March 1994.

[49] C. S. Jensen and R. Snodgrass. Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):954–974, December 1994.

[50] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying Temporal Models via a Conceptual Model. *Information Systems*, 19(7):513–547, October 1994.

[51] W. Käfer and H. Schöning. Mapping a Version Model to a Complex-Object Data Model. In *Proceedings of International Conference on Data Engineering*, pages 348–357, Tempe, Arizona, February 1992.

[52] W. Käfer and H. Schöning. Realizing a temporal complex-object data model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 266–275, San Diego, CA, June 1992.

[53] H. Kamp. Formal properties of 'now'. *Theoria*, 37(3):227–273, 1971.

[54] R. H. Katz. Towards a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 20(4):375–409, December 1990.

[55] W. Kim and H. T. Chou. Versions of Schema in OODB. In *Proceedings of the International Conference on Very Large Data Bases*, pages 148–159, Long Beach, CA, 1988.

[56] W. Kim, J. F. Garza, N. Ballou, and D. Woelk. Architecture of the ORION Next-Generation Database System. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):109–124, March 1990.

[57] N. Kline. Bibliography containing entries relevant to Temporal DBMS's. http://liinwww.ira.uka.de/bibliography/Database/time.html, January 27, 1996.

[58] E. McKenzie and R. Snodgrass. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[59] J. Melton (ed.). Database Language SQL. ISO/IEC JTC 1/SC21 WG 3 DBL:RIO-004 (ANSI TC X3H2-94-329), August 1994.

[60] J. Melton (ed.). Framework for SQL. ISO/IEC JTC 1/SC21 WG 3 DBL:RIO-003 (ANSI TC X3H2-94-328), August 1994.

[61] P. Øhrstrøm and P. F. V. Hasle. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Kluwer Academic Publishers, 1995.

[62] G. Özsoyoğlu and R. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513–532, August 1995.

[63] M. T. Özsu, R. Peters, D. Szafron, B. Irani, A. Lipka, and A. Munoz. TIGUKAT: A Uniform Behavioral Objectbase Management System. *VLDB Journal*, 4(3):445–492, January 1995.

[64] N. Pissinou, R. T. Snodgrass, R. Elmasri, I.S. Mumick, M.T. Özsu, B. Pernici, A. Segev, and B. Theodoulidis. Towards an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop. *ACM SIGMOD Records*, 23(1):35–51, March 1994.

[65] T. Plagemann, A. S. Saethre, and V. Goebel. Application Requirements and QoS Negotiation in Multimedia Systems. In *Proceedings of the 2nd International Workshop on Protocols for Multimedia Systems*, Salzburg, Austria, October 1995.

[66] A. Prior. *Past, Present and Future*. Oxford University Press, 1967.

[67] A. N. Prior. *Time and Modality*. Clarendon Press, Oxford, 1957.

[68] N. C. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, New York, 1971.

[69] E. Rose and A. Segev. TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints. In *Proceedings of the 10th International Conference on the Entity Relationship Approach*, San Mateo, California, October 1991.

[70] E. Rose and A. Segev. TOOA: A Temporal Object-Oriented Algebra. In *Proceedings of the European Conference on Object-Oriented Programming*, July 1993.

[71] E. Rose and A. Segev. TOOSQL – A Temporal Object-oriented Query Language. In *Proceedings of the 10th International Conference on the Entity-Relationship Approach*, Dallas, TX, 1993.

[72] L. A. Rowe and M. R. Stonebraker. The POSTGRES Data Model. In *Proceedings of the International Conference on Very Large Data Bases*, pages 83–96, 1987.

[73] R. v. B. Rucker. *Geometry, Relativity and The Fourth Dimension.* Dover Publications, Inc., New York, 1977.

[74] D. Schmidt, R. Marti, A. K. Dittrich, and W. Dreyer. Time Series, a Neglected Issue in Temporal Database Research? In Clifford and Tuzhlin [27], pages 214–234.

[75] E. Sciore. Multidimensional Versioning for Object-Oriented Databases. In *Proc. Second International Conf. on Deductive and Object-Oriented Databases*, December 1991.

[76] E. Sciore. Using annotations to support multiple kinds of versioning in an object-oriented database system. *ACM Transactions on Database Systems*, 16(3):417–438, September 1991.

[77] E. Sciore. Versioning and Configuration Management in an Object-Oriented Data Model. *VLDB Journal*, 3(1):77–106, 1994.

[78] A. Segev and R. Chandra. *Advanced Database Systems*, volume 759 of *Lecture Notes in Computer Science*, chapter 10: A Data Model for Time-Series Analysis. Springer Verlag, 1993.

[79] A. Segev, C. S. Jensen, and R. T. Snodgrass. Report on The 1995 International Workshop on Temporal Databases. *ACM SIGMOD Records*, 24(4), December 1995.

[80] A. Segev and A. Shoshani. Logical Modeling of Temporal Data. In U. Dayal and I. Traiger, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 454–466, San Francisco, CA, May 1987.

[81] A. Segev and A. Shoshani. *A Temporal Data Model Based on Time Sequences*, chapter 11, pages 248–270. In Tansel et al. [93], 1993.

[82] D.W. Shipman. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.

[83] B. Skjellaug. Temporal Data: Time and Relational Databases. Research Report 246, Department of Informatics, University of Oslo, April 1997. ISBN 82-7368-161-0.

[84] R. Snodgrass, editor. *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[85] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In S. Navathe, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 236–246, Austin, TX, May 1985.

[86] R. T. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[87] R. T. Snodgrass. *Modern Database Systems*, chapter 19: Temporal Object Oriented Databases: A Critical Comparison. ACM Press, New York, 1995.

[88] R. T. Snodgrass (editor). *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, 1995.

[89] M. Stonebraker, L. A. Rowe, and M. Hirohama. The Implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, March 1990.

[90] B. Stroustrup and M. A. Ellis. *The Annotated C++ Reference Manual.* Addison-Wesley, 1990.

[91] S. Y. W. Su and H. M. Chen. A Temporal Knowledge Representation Model OSAM*/T and Its Query Language OQL/T. In *Proceedings of the International Conference on Very Large Data Bases*, pages 431–442, Barcelona, Spain, September 1991.

[92] A. U. Tansel. Adding Time Dimension to Relational Model and Extending Relational Algebra. *Information Systems*, 11(4):343–355, 1986.

[93] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass (eds.). *Temporal Databases: Theory, Design, and Implementation.* Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, 1993.

[94] V. J. Tsotras and A. Kumar. An Update of the Temporal Database Bibliography. *ACM SIGMOD Records*, 25(1), March 1996.

[95] A. Tuzhlin and J. Clifford. On Periodicity in Temporal Databases. *Information Systems*, 20(8):619–639, 1995.

[96] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical Design for Temporal Databases with Multiple Granularities. ISSE-TR 94-111, ISSE Department, George Mason University, 1994.

[97] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with Granularity of Time in Temporal Databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991.

[98] G. T. J. Wuu and U. Dayal. A Uniform Model for Temporal Object-Oriented Databases. In F. Golshani, editor, *Proceedings of the International Conference on Data Engineering*, volume 8, pages 584–593, Los Alamitos, CA, February 1992. IEEE Computer Society Press.

[99] G. T. J. Wuu and U. Dayal. *A Uniform Model for Temporal and Versioned Object-oriented Databases*, chapter 10, pages 230–247. In Tansel et al. [93], 1993.